

# **Ice Sheet System Model 2011**

## **User Guide**

Authors:

**Mathieu Morlighem<sup>2,4</sup>**  
**Hélène Seroussi<sup>2,4</sup>**  
**Éric Larour<sup>1</sup>**  
**Éric Rignot<sup>2,3</sup>**

<sup>1</sup>Division 35, Thermal and Cryogenics Section,  
Mechanical Division, MS 157-316.  
Jet Propulsion Laboratory, 4800 Oak Grove Drive, Pasadena, CA 91109.

<sup>2</sup>Division 33, Radar Science and Engineering Section,  
Communications, Tracking and Radar Division, MS 157-316.  
Jet Propulsion Laboratory, 4800 Oak Grove Drive, Pasadena, CA 91109.

<sup>3</sup>University of California, Irvine  
Department of Earth System Science  
Croul Hall, Irvine, CA 92697-3100

<sup>4</sup>Laboratoire de Mécanique des sols, Structures et matériaux (MSSMat)  
École Centrale Paris, CNRS UMR 8579  
Grande Voie des Vignes, 92295 Chtenay-Malabry Cedex, FRANCE  
November 7, 2011

## Summary

This manual explains how to use the ISSM code, capable of modeling ice flow in 2d and 3d using finite elements. Key-words: ISSM, ice flow, finite elements, 2-D, 3-D, matlab

# Contents

<b>1 Installation</b>	<b>7</b>
1.1 Download ISSM . . . . .	7
1.1.1 Requirements . . . . .	7
1.1.2 JPL users with an account on Huey . . . . .	7
1.1.3 Request the last stable version of ISSM . . . . .	7
1.2 Installation of ISSM . . . . .	8
1.2.1 Environment variables . . . . .	8
1.2.2 External packages installation . . . . .	8
1.2.3 Development packages installation . . . . .	8
1.2.4 ISSM compilation . . . . .	9
1.2.5 Installation on Windows 32 bits . . . . .	9
1.2.5.1 Cygwin . . . . .	9
1.2.5.2 Intel . . . . .	9
1.2.5.3 Microsoft Visual Studio Express . . . . .	9
1.2.5.4 Cygwin and intel compilers . . . . .	9
<b>2 Model Creation</b>	<b>11</b>
2.1 "startup.m" file . . . . .	11
2.2 Model class . . . . .	11
2.2.1 Matlab's model object . . . . .	11
2.2.2 Saving/loading a model . . . . .	12
2.3 Model fields . . . . .	12
2.3.1 Mesh properties . . . . .	12
2.3.2 Mask parameters . . . . .	13
2.3.3 Geometry properties . . . . .	14
2.3.4 Constant parameters . . . . .	14
2.3.5 Surface forcing parameters . . . . .	14
2.3.6 Basal forcing parameters . . . . .	14
2.3.7 Material parameters . . . . .	15
2.3.8 Friction parameters . . . . .	15
2.3.9 Flow equation parameters . . . . .	15
2.3.10 Time stepping parameters . . . . .	16
2.3.11 Initialization fields . . . . .	16
2.3.12 Rifts parameters . . . . .	16
2.3.13 Debugging parameters . . . . .	16
2.3.14 Verbose parameters . . . . .	17
2.3.15 Setting parameters . . . . .	17
2.3.16 Solver parameters . . . . .	17
2.3.17 Cluster parameters . . . . .	17
2.3.18 Balance thickness solution parameters . . . . .	17
2.3.19 Diagnostic solution parameters . . . . .	18
2.3.20 Grounding line migration parameters . . . . .	18
2.3.21 Hydrology solution parameters . . . . .	19
2.3.22 Prognostic solution parameters . . . . .	19
2.3.23 Thermal solution parameters . . . . .	19

2.3.24	Steady-state solution parameters . . . . .	19
2.3.25	Transient solution parameters . . . . .	20
2.3.26	Automatic differentiation parameters . . . . .	20
2.3.27	FLAIM (Flight Line Adaptation using Ice Sheet Model) parameters . . . . .	20
2.3.28	Inversion parameters . . . . .	20
2.3.29	Sensitivity analysis parameters . . . . .	21
2.3.30	Results . . . . .	22
2.3.31	Radar overlay parameters . . . . .	22
2.3.32	Miscellaneous . . . . .	22
<b>3</b>	<b>Mesh</b>	<b>23</b>
3.1	Domain outline . . . . .	23
3.2	Trimesh Mesher . . . . .	23
3.3	Bamg Mesher . . . . .	24
3.3.1	Isotropic mesh generation . . . . .	24
3.3.1.1	Domain . . . . .	24
3.3.1.2	hmin/hmax . . . . .	24
3.3.1.3	hVertices . . . . .	24
3.3.2	Anisotropic mesh generation . . . . .	24
3.3.2.1	field/err . . . . .	24
3.3.2.2	gradation . . . . .	24
3.3.2.3	anisomax . . . . .	25
<b>4</b>	<b>Model parameterization</b>	<b>26</b>
4.1	Mask definition . . . . .	26
4.2	Parameter file . . . . .	26
4.2.1	Interpolation routines . . . . .	26
4.2.2	Example . . . . .	27
4.3	Verbosity levels . . . . .	28
4.4	Model extrusion (optional) . . . . .	28
4.5	Ice flow approximations . . . . .	29
<b>5</b>	<b>Solutions</b>	<b>31</b>
5.1	Launching a solution sequence . . . . .	31
5.2	Parallel computing . . . . .	31
5.2.1	Setting up the environment to use the parallel mode . . . . .	31
5.2.2	password-less SSH login . . . . .	31
5.2.2.1	SSH passthrough . . . . .	32
5.2.2.2	Tunneling . . . . .	32
5.2.3	FAQ . . . . .	32
<b>6</b>	<b>Advanced features</b>	<b>33</b>
6.1	Inverse methods . . . . .	33
6.1.1	Objective functions . . . . .	33
6.1.1.1	Absolute misfit . . . . .	34
6.1.1.2	Relative misfit . . . . .	34
6.1.1.3	Logarithmic misfit . . . . .	34
6.1.1.4	Thickness misfit . . . . .	34
6.1.1.5	Drag gradient . . . . .	34
6.1.1.6	Thickness gradient . . . . .	35
6.1.2	Inversion parameters . . . . .	35
6.1.2.1	Control parameters . . . . .	35
6.1.2.2	Constraints . . . . .	35
6.1.2.3	Optimization parameters . . . . .	35
6.1.3	Launching a control method . . . . .	36
6.2	Rifts . . . . .	36
6.2.1	Rifts creation . . . . .	36

6.2.2	Rifts tip refining . . . . .	36
6.2.3	Rifts in parameter file . . . . .	37
6.2.4	Solving for rifts . . . . .	37
6.2.5	Rifts plotting . . . . .	37
6.2.6	Rifts when using Yams mesh adaptation . . . . .	37
6.2.7	Adding rifts to an existing mesh . . . . .	37
6.3	Quantifications of Margins and Uncertainties with Dakota . . . . .	38
<b>7</b>	<b>Plotting</b>	<b>40</b>
7.1	Standard plots . . . . .	40
7.2	Quiver plots . . . . .	40
7.2.1	ColorLevels . . . . .	41
7.2.2	Scaling . . . . .	41
7.2.3	Autoscale . . . . .	42
7.2.4	Density . . . . .	42
7.3	Section plots . . . . .	42
7.3.1	Section plots options . . . . .	43
7.3.1.1	Resolution . . . . .	43
7.3.1.2	Show section . . . . .	44
7.4	Special plots . . . . .	44
7.4.1	basaldrag . . . . .	44
7.4.2	boundaries . . . . .	44
7.4.3	boundariesrift . . . . .	45
7.4.4	driving_stress . . . . .	45
7.4.5	elementnumbering . . . . .	45
7.4.6	elements_type . . . . .	46
7.4.7	gridnumbering . . . . .	46
7.4.8	highlightelements . . . . .	47
7.4.9	highlightgrids . . . . .	47
7.4.10	mesh . . . . .	48
7.4.11	pressureload . . . . .	48
7.4.12	riftfraction . . . . .	48
7.4.13	riftpenetration . . . . .	49
7.4.14	riftrlevel . . . . .	49
7.4.15	rifts . . . . .	50
7.4.16	riftvvel . . . . .	50
7.5	Plot options . . . . .	51
7.5.1	Axis . . . . .	51
7.5.1.1	axis . . . . .	51
7.5.1.2	antzoom . . . . .	51
7.5.1.3	view . . . . .	51
7.5.1.4	xlim . . . . .	51
7.5.1.5	ylim . . . . .	51
7.5.1.6	zlim . . . . .	52
7.5.2	Title and labels . . . . .	52
7.5.2.1	title . . . . .	52
7.5.2.2	fontsize . . . . .	52
7.5.2.3	fontweight . . . . .	52
7.5.2.4	xlabel . . . . .	52
7.5.2.5	ylabel . . . . .	52
7.5.2.6	text . . . . .	52
7.5.2.7	textposition . . . . .	52
7.5.2.8	textsize . . . . .	52
7.5.2.9	textweight . . . . .	52
7.5.2.10	textcolor . . . . .	53
7.5.3	Color bars . . . . .	53
7.5.3.1	caxis . . . . .	53

---

7.5.3.2	colorbar	53
7.5.3.3	colorbarpos	54
7.5.3.4	colormap	54
7.5.3.5	log	54
7.5.3.6	wrapping	55
7.5.4	Contours	55
7.5.4.1	contourlevels	55
7.5.4.2	contourticks	56
7.5.4.3	contouronly	56
7.5.5	Latitude/Longitude	57
7.5.5.1	latlon	57
7.5.5.2	latlonnumbering	57
7.5.5.3	latlonclick	58
7.5.6	Radar overlay	58
7.5.6.1	overlay	58
7.5.6.2	alpha	58
7.5.6.3	highres	59
7.5.7	Other options	59
7.5.7.1	streamlines	59
7.5.7.2	edgecolor	59
7.5.7.3	smooth	60
7.5.7.4	expdisp	60
7.5.7.5	expstyle	60
7.5.7.6	mask	61
7.5.7.7	northarrow	61
7.5.7.8	scaleruler	61
<b>8</b>	<b>Miscellaneous Tools</b>	<b>62</b>
8.0.8	Mesh	62
8.0.9	Model parameterization	62
8.0.10	Mask	63
8.0.11	Interpolation	63
8.0.12	Argus files	63
8.0.13	Results analysis	63
<b>9</b>	<b>Tutorial</b>	<b>64</b>
9.1	Pine Island Glacier tutorial	64
9.2	Radar image	64
9.3	Mesh creation	65
9.3.1	Creation of the domain outline	65
9.3.2	Mesh generation	66
9.3.3	Anisotropic mesh adaptation	67
9.4	Geography	68
9.4.1	Building the 'Iceshelves.exp' file	68
9.4.2	Building the 'Island.exp' file	68
9.4.3	Geography command	69
9.5	Parameter file	69
9.5.1	Parameterization command	70
9.5.2	Extrusion (optional)	70
9.5.3	Setting elements type	71
9.6	Wrap up	71
9.7	Launching the diagnostic solution sequence	72
<b>10</b>	<b>FAQ</b>	<b>73</b>
10.1	The following message appears in the errlog file when launching my job in parallel:	73
10.2	How to debug a matlab crash in serial mode?	73
10.3	How to debug a parallel solution crash?	73

# Chapter 1

## Installation

### 1.1 Download ISSM

#### 1.1.1 Requirements

ISSM has been compiled and tested on many LINUX/UNIX based operating system. It can be installed on the following OS:

- Linux 32/64
- Mac OS X
- Windows XP (with [cygwin](#) or equivalent)

#### 1.1.2 JPL users with an account on Huey

ISSM is actively managed using a code versioning system called [SVN](#). The code is stored in a repository, and can be remotely fetched, modified, updated, and uploaded. This allows for multiple users to develop the code in an organized way. Users with an account on [huey.jpl.nasa.gov](#) can directly download the code from the repository. In order to fetch a version of the code, users will need to install [SVN](#) on their machine. Once [SVN](#) has been installed, ISSM can be downloaded by the following command:

```
$ svn --username your_user_name checkout http://s383-rhat.jpl.nasa.gov/issm/svn/issm/trunk
```

where `your_user_name` is the user name on s383.

This command will download the last version of the code ISSM from the repository, onto the current local directory. Users are free to choose whichever location they want and to rename issm.

#### 1.1.3 Request the last stable version of ISSM

Non-JPL users willing to collaborate on a project using ISSM can request more information to [Eric Larour](#).

## 1.2 Installation of ISSM

### 1.2.1 Environment variables

The compilation of ISSM requires several environment variables. Add the following lines in your shell environment script: .bashrc

```
#ISSM
export ISSM_TIER=ISSMPATH
export ISSM_ARCH=ARCH
source $ISSM_TIER/etc/environment.sh

.cshrc

#ISSM
setenv ISSM_TIER ISSMPATH
setenv ISSM_ARCH ARCH
source $ISSM_TIER/etc/environment.csh
```

Where ISSMPATH is the path of ISSM main directory (ex: /home/user1/svn/issm/trunk) and ARCH is the system architecture (ex: linux-gnu-amd64, macosx-gnu,...).

### 1.2.2 External packages installation

All ISSM external packages are located in the directory `externalpackages` of the trunk. Several packages may be installed depending on what users want to do. At least the following packages must be installed:

- mpich2 (to be installed first)
- matlab
- petsc
- metis
- triangle

For each library, a directory `configs` may exist. In this directory, the user will find subdirectories with different machine names. All the content of the subdirectory corresponding to the given machine has to be copied. For example:

```
$ cd $ISSM_TIER/externalpackages/mpich2
$ cp configs/astrid/* .
$ ./install.sh
```

There is no guarantee the compilation will work on all systems. A lot of tweaking of the install.sh files will probably be involved. Especially, the configuration part of the install. (See [compilation troubleshooting](#)) Note: matlab is only a softlink to the actual matlab directory

### 1.2.3 Development packages installation

ISSM relies on autotools to make source-code packages portable to many Unix-like systems. To install all development packages, use the same methods:

```
$ cd $ISSM_TIER/devpackages/autoconf
$ ./install.sh
$ cd $ISSM_TIER/devpackages/automake
$ ./install.sh
```

### 1.2.4 ISSM compilation

Last step: generate the Makefiles needed to compile ISSM. First, ISSM must be reconfigured:

```
$ cd $ISSM_TIER
$ ./scripts/automakererun.sh
```

ISSM can then be configured for the given OS. Several scripts exist to configure ISSM. For example:

```
$ ./configs/astrid/configure.sh
```

For other platforms, one might need to write its own configuration file.

ISSM can now be compiled:<p/>

```
$ cd $ISSM_TIER
$ make
$ make install
```

ISSM installation is done!

### 1.2.5 Installation on Windows 32 bits

#### 1.2.5.1 Cygwin

Users should first download the following the cygwin package, which emulates unix behaviour on windows machines. The software can be found at: <http://www.cygwin.com>. The standard install is sufficient, provided users add the following packages: openssh, subversion, autotools, automake and make. Users can adapt the cygwin install if any other routine is not available in the initial install.

#### 1.2.5.2 Intel

Users should then download the Intel C/C++ compiler (trial version is ok, but for permanent use of ISSM in development mode, a permanent intel license will be necessary) at the following webpage: <http://software.intel.com/en-us/intel-compilers/>. The only compiler necessary is the C++ compiler for Windows 32 bits. Matlab support only extends to the family of Intel compilers (Borland or Myngw compilers are not supported), which is why we adopted this compiler.

#### 1.2.5.3 Microsoft Visual Studio Express

Users should also download Microsoft Visual Studio, necessary for the Intel C++ compiler to correctly link with the existing system libraries of the Windows XP platform. The web page is: <http://microsoft.com/express/>. A free version is available.

#### 1.2.5.4 Cygwin and intel compilers

Before even trying to install ISSM, users should make sure the icl (Intel C++) compiler is working correctly on a simple hello.cpp program, such as:

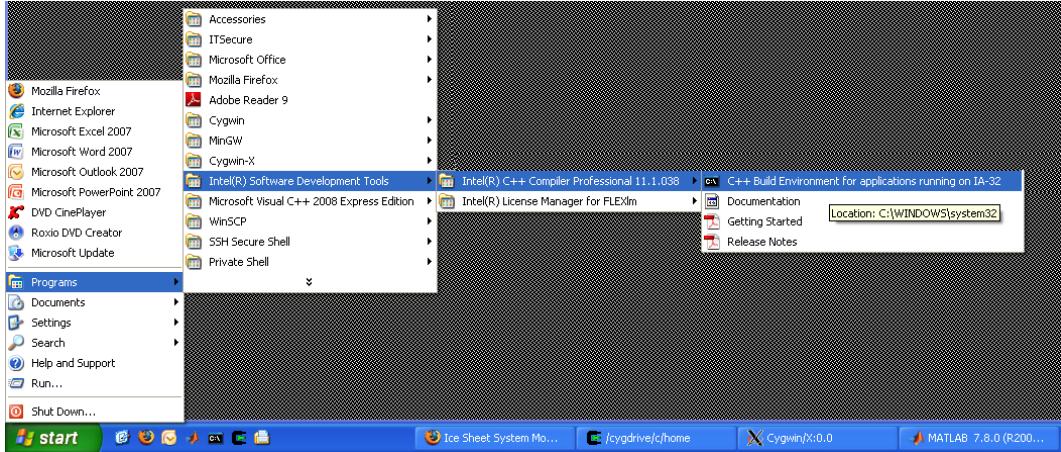
```
#include "stdio.h"
int main(){
printf("hello\n");
return 1;
}
```

Part of the problem is to get the intel compiler icl to link using the microsoft visual studio. For example, for a VC9 installation, the following has to be included in the C/Program Files/Intel/Compiler/11.1/026/bin/ia32/icl file:

```
-Qlocation,link,"C:\Program Files\Microsoft Visual Studio 9.0\VC\bin"
-Qprec-div -Qprec-sqrt -Qansi_alias
}
```

Once this is done, icl should compile fine.

Once the hello program works, users need to provide the environment variables necessary to cygwin to correctly recognize the icl compiler and compile. This phase is tricky, as only the command line tool of Intel is capable of setting the correct environment variables. One way of figuring out which variables are set by Intel and Microsoft Visual Studio is to launch the command line intel tool, as described in the following picture:



Then use the command line from intel to go to the Cygwin installation directory, and run the `./Cygwin.bat` batch file. This will boot the cygwin environment, while at the same time keeping the settings from the intel command line tool. A simple export command at the command prompt will then inform users of which variables have been set. The file `ISSM_TIER/externalpackages/intel/intel.sh` is an example of the environment variables specific to the icl compiler. Once the intel compiler is correctly installed, the process of compiler ISSM is very similar to the linux installs. Just use the `config/win32/win32.sh` file to configure ISSM. Then compile the `ISSM_TIER/src/c/` code by typing:

```
$ make
```

This stage will then error out, as automake is not capable of configuring the Makefiles in a way that is compatible with the icl compiler. Just run:

```
$ source ./intel-make.sh
```

to complete the process. You can then compile the serial modules in `ISSM_DIR/src/mex/` by doing:

```
$ make install
```

# Chapter 2

## Model Creation

### 2.1 "startup.m" file

When matlab is launched, all the ISSM tools need to be correctly located. This is done by the `startup.m` file located in the `trunk/` directory. This file needs to be located in the starting directory of matlab, or in the directory where matlab will establish its root. When matlab is launch, the following message should appear:

```
To get started with ISSM, type issmdoc at the command prompt.
```

If not, `startup.m` has not been found by matlab when it started and ISSM will not work properly. A simple way to avoid having a `startup.m` in every working directory is to add the following alias to the `~/.bashrc`:

```
alias matlab='matlab -r "addpath $ISSM_TIER; startup;"'
```

The `startup.m` file will be automatically added and executed whenever matlab is launched.

### 2.2 Model class

#### 2.2.1 Matlab's model object

All the data belonging to a model (geometry, node coordinates, results,...) is held in the same Matlab object `model`. To create a new model, one can type the following command in Matlab's Command window:

```
>> md=model;
```

This will create a new model named "md" whose class is "model". The information contained in the model "md" are grouped bby class, that contain fields related to a particular aspect of the model: mesh, materials' properties, friction, diagnostic solution, results of the runs, etc... When one creates a new model, all these fields are empty or NaN (not a number), but "md" is ready to be used as a model. The list of these classes is displayed when typing:

```
>> md
md =

    mesh : [1x1 mesh]          -- mesh properties
    mask : [1x1 mask]          -- defines grounded and floating elements
    geometry : [1x1 geometry]   -- surface elevation, bedrock topography, ice thickness.
    constants : [1x1 constants] -- physical constants
    surfaceforcings : [1x1 surfaceforcings] -- surface forcings
    basalforcings : [1x1 basalforcings] -- bed forcings
    materials : [1x1 materials] -- material properties
```

```

        friction : [1x1 friction]           -- basal friction/drag properties
        flowequation : [1x1 flowequation]    -- flow equations
        timestepping : [1x1 timestepping]    -- time stepping for transient models
        initialization : [1x1 initialization]
            rifts : [1x1 rifts]             -- rifts properties
            debug : [1x1 debug]             -- debugging tools (valgrind, gprof)
            verbose : [1x1 verbose]          -- verbosity level in solve
            settings : [1x1 settings]        -- settings properties
            solver : [1x1 solver]            -- PETSc options for each solution
            cluster : [1x1 none]             -- cluster parameters (number of cpus...)
        balancethickness : [1x1 balancethickness] -- parameters for balancethickness solution
            diagnostic : [1x1 diagnostic]     -- parameters for diagnostic solution
        groundingline : [1x1 groundingline]      -- parameters for groundingline solution
            hydrology : [1x1 hydrology]       -- parameters for hydrology solution
            prognostic : [1x1 prognostic]      -- parameters for prognostic solution
            thermal : [1x1 thermal]           -- parameters for thermal solution
        steadystate : [1x1 steadystate]        -- parameters for steadystate solution
            transient : [1x1 transient]       -- parameters for transient solution
            autodiff : [1x1 autodiff]          -- automatic differentiation parameters
            flaim : [1x1 flaim]              -- flaim parameters
        inversion : [1x1 inversion]           -- parameters for inverse methods
            qmu : [1x1 qmu]                 -- dakota properties
            results : [1x1 struct]            -- model results
        radaroverlay : [1x1 radaroverlay]      -- radar image for plot overlay
        miscellaneous : [1x1 miscellaneous]   -- miscellaneous fields

```

One can add a note and/or a name to the model in order to remember easily what it deals with:

```
>> md.miscellaneous.name='PineIslandGlacier';
>> md=addnote(md,'Pine Island Glacier test 1, geometry of 1996');
```

## 2.2.2 Saving/loading a model

One can save the model with all its fields so that the saved file contains all the information in the model, type the following command:

```
>> save square.model md
```

That will create a file `square.model` made from the model `md`. To load this file, type:

```
>> loadmodel square.model
```

the loaded model will be named `md`.

## 2.3 Model fields

### 2.3.1 Mesh properties

One can display all these fields by typing:

```
>> md.mesh
```

- `md.mesh.numberofelements`: number of elements
- `md.mesh.numberofvertices`: number of vertices
- `md.mesh.elements`: index into (x,y,z), coordinates of the vertices
- `md.mesh.x`: vertices x coordinate
- `md.mesh.y`: vertices y coordinate

- `md.mesh.z`: vertices z coordinate
- `md.mesh.edges`: edges of the 2d mesh (vertex1 vertex2 element1 element2)
- `md.mesh.numberofedges`: number of edges of the 2d mesh
- `md.mesh.dimension`: mesh dimension (2d or 3d)
- `md.mesh.numberoflayers`: number of extrusion layers
- `md.mesh.vertexonbed`: lower vertices flags list
- `md.mesh.elementonbed`: lower elements flags list
- `md.mesh.vertexonsurface`: upper vertices flags list
- `md.mesh.elementonsurface`: upper elements flags list
- `md.mesh.uppervertex`: upper vertex list (NaN for vertex on the upper surface)
- `md.mesh.upperelements`: upper element list (NaN for element on the upper layer)
- `md.mesh.lowervertex`: lower vertex list (NaN for vertex on the lower surface)
- `md.mesh.lowerelements`: lower element list (NaN for element on the lower layer)
- `md.mesh.vertexonboundary`: vertices on the boundary of the domain flag list
- `md.mesh.segments`: edges on domain boundary (vertex1 vertex2 element)
- `md.mesh.segmentmarkers`: number associated to each segment
- `md.mesh.vertexconnectivity`: list of vertices connected to vertex i
- `md.mesh.elementconnectivity`: list of vertices connected to element i
- `md.mesh.average_vertex_connectivity`: average number of vertices connected to one vertex
- `md.mesh.extractedvertices`: vertices extracted from the model
- `md.mesh.extractedelements`: elements extracted from the model
- `md.mesh.lat`: vertices latitude
- `md.mesh.long`: vertices longitude
- `md.mesh.hemisphere`: Indicate hemisphere 'n' or 's'gg

### 2.3.2 Mask parameters

One can display all these fields by typing:

```
>> md.mask
```

- `md.mask.elementonfloatingice`: element on floating ice flags list
- `md.mask.vertexonfloatingice`: vertex on floating ice flags list
- `md.mask.elementongroundedice`: element on grounded ice list
- `md.mask.vertexongroundedice`: vertex on grounded ice flags list
- `md.mask.elementonwater`: element on water flags list
- `md.mask.vertexonwater`: vertex on water flags list

### 2.3.3 Geometry properties

One can display all these fields by typing:

```
>> md.geometry
```

- `md.geometry.surface`: surface elevation
- `md.geometry.thickness`: ice thickness
- `md.geometry.bed`: bed elevation
- `md.geometry.bathymetry`: bathymetry elevation
- `md.geometry.hydrostatic_ratio`: coefficient for ice shelves' thickness correction: hydrostatic ratio H obs + (1-hydrostatic ratio) H hydro

### 2.3.4 Constant parameters

One can display all these fields by typing:

```
>> md.constants
```

- `md.constants.g`: gravitational acceleration
- `md.constants.yts`: number of seconds in a year
- `md.constants.referenceTemperature`: reference temperature used in the enthalpy model

### 2.3.5 Surface forcing parameters

One can display all these fields by typing:

```
>> md.surfaceforcings
```

- `md.surfaceforcings.accumulation_rate`: surface accumulation rate (in m)
- `md.surfaceforcings.ablation_rate`: surface ablation rate (in m)
- `md.surfaceforcings.mass_balance`: surface mass balance (in m)

### 2.3.6 Basal forcing parameters

One can display all these fields by typing:

```
>> md.basalforcings
```

- `md.basalforcings.melting_rate`: basal melting rate (positive if melting)
- `md.basalforcings.melting_rate_correction`: additional melting applied when the grounding line retreats
- `md.basalforcings.geothermalflux`: geothermal heat flux (in W/m<sup>2</sup>)

### 2.3.7 Material parameters

One can display all these fields by typing:

```
>> md.materials
```

- `md.materials.rho_ice`: ice density (in kg/m<sup>3</sup>)
- `md.materials.rho_water`: water density (in kg/m<sup>3</sup>)
- `md.materials.heatcapacity`: heat capacity (in J/kg/K)
- `md.materials.thermalconductivity`: ice thermal conductivity (in W/m/K)
- `md.materials.meltingpoint`: melting point of ice at 1atm in K
- `md.materials.latentheat`: latent heat of fusion (in J/m<sup>3</sup>)
- `md.materials.beta`: rate of change of melting point with pressure (in K/Pa)
- `md.materials.mixed_layer_capacity`: mixed layer capacity (in W/kg/K)
- `md.materials.thermal_exchange_velocity`: thermal exchange velocity (in m/s)
- `md.materials.rheology_B`: flow law parameter (in Pa/s<sup>1/n</sup>)
- `md.materials.rheology_n`: Glen's flow law exponent
- `md.materials.rheology_law`: law for the temperature dependance of the rheology: 'None', 'Paterson' or 'Arrhenius'

### 2.3.8 Friction parameters

The friction is defined as:

$$\tau_b = -k^2 N^r \|v\|^{s-1} v_b \quad (2.1)$$

One can display all these fields by typing:

```
>> md.friction
```

- `md.friction.coefficient`: friction coefficient (in IS)
- `md.friction.p`: p exponent
- `md.friction.q`: q exponent

### 2.3.9 Flow equation parameters

One can display all these fields by typing:

```
>> md.flowequation
```

- `md.flowequation.ismacayealpattyn`: is the macayeal or pattyn approximation used ?
- `md.flowequation.ishutter`: is the shallow ice approximation used ?
- `md.flowequation.isstokes`: are the Full-Stokes equations used ?
- `md.flowequation.vertex_equation`: flow equation for each vertex
- `md.flowequation.element_equation`: flow equation for each element
- `md.flowequation.bordermacayeal`: vertices on MacAyeal's border (for tiling)
- `md.flowequation.borderpattyn`: vertices on Pattyn's border (for tiling)
- `md.flowequation.borderstokes`: vertices on Stokes' border (for tiling)

### 2.3.10 Time stepping parameters

One can display all these fields by typing:

```
>> md.timestepping
```

- `md.timestepping.time_step`: length of time steps (in yrs)
- `md.timestepping.final_time`: final time to stop the simulation (in yrs)
- `md.timestepping.time_adapt`: use cfl condition to define time step ? (0 or 1)
- `md.timestepping.cfl_coefficient`: coefficient applied to cfl condition

### 2.3.11 Initialization fields

One can display all these fields by typing:

```
>> md.initialization
```

- `md.initialization.vx`: x component of velocity
- `md.initialization.vy`: y component of velocity
- `md.initialization.vz`: z component of velocity
- `md.initialization.vel`: velocity norm
- `md.initialization.pressure`: pressure field
- `md.initialization.temperature`: temperature in Kelvins
- `md.initialization.watercolumn`: thickness of subglacial water
- `md.initialization.waterfraction`: fraction of water in the ice

### 2.3.12 Rifts parameters

One can display all these fields by typing:

```
>> md.rifts
```

- `md.rifts.numrifts`: number of rifts
- `md.rifts.riftstruct`: structure containing all rift information (vertices coordinates, segments, type of melange, ...)
- `md.rifts.riftproperties`

### 2.3.13 Debugging parameters

One can display all these fields by typing:

```
>> md.debug
```

- `md.debug.valgrind`: use Valgrind to debug (0 or 1)
- `md.debug.gprof`: use gnu-profiler find out where the time is spent

### 2.3.14 Verbose parameters

One can display all these fields by typing:

```
>> md.verbose
```

- `md.verbose.mprocessor`: display messages concerning processor
- `md.verbose.module`: display messages concerning modules
- `md.verbose.solution`: display messages concerning solution
- `md.verbose.solver`: display messages concerning solver
- `md.verbose.convergence`: display messages concerning solution convergence
- `md.verbose.control`: display messages concerning inversion

### 2.3.15 Setting parameters

One can display all these fields by typing:

```
>> md.settings
```

- `md.settings.io_gather`: I/O gathering strategy for result outputs (default 1)
- `md.settings.lowmem`: is the memory limited ? (0 or 1)
- `md.settings.results_on_vertices`: provide results on vertices instead of patch (0 or 1)
- `md.settings.output_frequency`: frequency at which results are saved in all solutions with multiple time steps
- `md.settings.waitonlock`: maximum number of minutes to wait for batch results, or return 0

### 2.3.16 Solver parameters

One can display all these fields by typing:

```
>> md.solver
```

### 2.3.17 Cluster parameters

One can display all these fields by typing:

```
>> md.cluster
```

### 2.3.18 Balance thickness solution parameters

One can display all these fields by typing:

```
>> md.balancethickness
```

- `md.balancethickness.spcthickness`: thickness constraints (NaN means no constraint)
- `md.balancethickness.thickening_rate`: ice thickening rate used in the mass conservation ( $dh/dt$ )
- `md.balancethickness.stabilization`: 0->no, 1->Artificial diffusivity, 3->discontinuous Galerkin

### 2.3.19 Diagnostic solution parameters

One can display all these fields by typing:

```
>> md.diagnostic
```

- `md.diagnostic.restol`: mechanical equilibrium residue convergence criterion
- `md.diagnostic.reltol`: velocity relative convergence criterion, NaN -> not applied
- `md.diagnostic.abstol`: velocity absolute convergence criterion, NaN -> not applied
- `md.diagnostic.maxiter`: maximum number of nonlinear iterations
- `md.diagnostic.viscosity_overshoot`: over-shooting constant defined as:

$$\mu^{\text{new}} = \mu^{\text{new}} + \alpha (\mu^{\text{new}} - \mu^{\text{old}}) \quad (2.2)$$

- `md.diagnostic.spcvx`: x-axis velocity constraint (NaN means no constraint)
- `md.diagnostic.spcvy`: y-axis velocity constraint (NaN means no constraint)
- `md.diagnostic.spcvz`: z-axis velocity constraint (NaN means no constraint)
- `md.diagnostic.icefront`: segments on ice front list
- `md.diagnostic.rift_penalty_threshold`: threshold for instability of mechanical constraints
- `md.diagnostic.rift_penalty_lock`: number of iterations before rift penalties are locked
- `md.diagnostic.penalty_factor`: offset used by penalties:

$$\kappa = 10^{\text{penalty\_offset}} \max_{i,j} |K_{ij}| \quad (2.3)$$

- `md.diagnostic.vertex_pairing`: pairs of vertices that are penalized
- `md.diagnostic.shelf_dampening`: use dampening for floating ice ? Only for Stokes model
- `md.diagnostic.referential`: local referential
- `md.diagnostic.requested_outputs`: additional outputs requested

### 2.3.20 Grounding line migration parameters

One can display all these fields by typing:

```
>> md.groundingline
```

- `md.groundingline.migration`: type of grounding line migration: 'SoftMigration', 'AggressiveMigration' or 'None'
- `md.groundingline.melting_rate`: melting rate applied when previously grounded parts start floating

### 2.3.21 Hydrology solution parameters

One can display all these fields by typing:

```
>> md.hydrology
```

- `md.hydrology.spcwatercolumn`: water thickness constraints (NaN means no constraint)
- `md.hydrology.n`: Manning roughness coefficient
- `md.hydrology.CR`: turtuosity parameter
- `md.hydrology.p`: dimensionless exponent in Manning velocity formula
- `md.hydrology.q`: dimensionless exponent in Manning velocity formula
- `md.hydrology.kn`: parameter in effective pressure formula
- `md.hydrology.stabilization` : artificial diffusivity (default is 1)

### 2.3.22 Prognostic solution parameters

One can display all these fields by typing:

```
>> md.prognostic
```

- `md.prognostic.spcthickness`: thickness constraints (NaN means no constraint)
- `md.prognostic.hydrostatic_adjustment`: adjustment of ice shelves surface and bed elevations: 'Incremental' or 'Absolute'
- `md.prognostic.stabilization`: 0->no, 1->Artificial diffusivity, 3->Discontinuous Galerkin
- `md.prognostic.penalty_factor`: offset used by penalties

$$\kappa = 10^{\text{penalty\_offset}} \max_{i,j} |K_{ij}| \quad (2.4)$$

- `md.prognostic.vertex_pairing`: pairs of vertices that are penalized

### 2.3.23 Thermal solution parameters

One can display all these fields by typing:

```
>> md.thermal
```

- `md.thermal.spctemperature`: temperature constraints (NaN means no constraint)
- `md.thermal.stabilization`: 0: no, 1: artificial diffusivity, 2: SUPG
- `md.thermal.maxiter`: maximum number of non linear iterations
- `md.thermal.penalty_lock`: stabilize unstable thermal constraints that keep zigzagging after n iteration (default is 0, no stabilization)
- `md.thermal.penalty_threshold`: threshold to declare convergence of thermal solution (default is 0)

### 2.3.24 Steady-state solution parameters

One can display all these fields by typing:

```
>> md.steadystate
```

- `md.steadystate.reltol`: relative tolerance criterion
- `md.steadystate.maxiter`: maximum number of iterations
- `md.steadystate.requested_outputs`: additional requested outputs

### 2.3.25 Transient solution parameters

One can display all these fields by typing:

```
>> md.transient
```

- `md.transient.isprognostic`: indicates if a prognostic solution is used in the transient
- `md.transient.isthermal`: indicates if a thermal solution is used in the transient
- `md.transient.isdiagnostic`: indicates if a diagnostic solution is used in the transient
- `md.transient.isgroundingline`: indicates if a groundingline migration is used in the transient

### 2.3.26 Automatic differentiation parameters

One can display all these fields by typing:

```
>> md.autodiff
```

- `md.autodiff.isautodiff`: indicates if the automatic differentiation is activated
- `md.autodiff.forward`: forward differentiation
- `md.autodiff.reverse`: backward differentiation

### 2.3.27 FLAIM (Flight Line Adaptation using Ice Sheet Model) parameters

One can display all these fields by typing:

```
>> md.flaim
```

- `md.flaim.targets`: name of kml targets file
- `md.flaim.tracks`: name of kml tracks file
- `md.flaim.flightreqs`: structure of kml flight requirements
- `md.flaim.criterion`: element or nodal criterion for flight path evaluation
- `md.flaim.gridsatequator`: number of grids at equator (determines resolution)
- `md.flaim.usevalueordering`: flag to consider target values for flight path evaluation
- `md.flaim.split_antimeridian`: flag to split polygons on the antimeridian
- `md.flaim.solution`: name of kml solution file
- `md.flaim.quality`: quality of kml solution

### 2.3.28 Inversion parameters

One can display all these fields by typing:

```
>> md.inversion
```

- `md.inversion.iscontrol`: is inversion activated?
- `md.inversion.control_parameters`: parameter where inverse control is carried out; ex: `{'FrictionCoefficient'}` or `{'RheologyBbar'}`
- `md.inversion.cost_functions_coefficients`: cost functions coefficients applied to the misfit of each vertex and for each control parameter
- `md.inversion.nsteps`: number of optimization searches

- `md.inversion.maxiter_per_step`: maximum iterations during each optimization step
- `md.inversion.cost_functions`: indicate the type of response for each optimization steps
- `md.inversion.cost_function_threshold`: misfit convergence criterion. Default is 1
- `md.inversion.gradient_scaling`: scaling factor on gradient direction during optimization, for each optimization step
- `md.inversion.step_threshold`: decrease threshold for misfit, default is 30
- `md.inversion.min_parameters`: absolute minimum acceptable value of the inversed parameter on each vertex
- `md.inversion.max_parameters`: absolute maximum acceptable value of the inversed parameter on each vertex
- `md.inversion.gradient_only`: stop control method solution at gradient
- `md.inversion.num_control_parameters`: number of control parameters
- `md.inversion.num_cost_functions`: number of cost functions
- `md.inversion.vx_obs`: observed velocity x component (in m/a)
- `md.inversion.vy_obs`: observed velocity y component (in m/a)
- `md.inversion.vel_obs`: observed velocity magnitude (in m/a)
- `md.inversion.thickness_obs`: observed thickness (in m)

### 2.3.29 Sensitivity analysis parameters

One can display all these fields by typing:

```
>> md.qmu
```

- `md.qmu.isdakota`: is qmu analysis activated?
- `md.qmu.variables`
- `md.qmu.responses`
- `md.qmu.numberofresponses`: number of responses
- `md.qmu.qmu_params`
- `md.qmu.results`
- `md.qmu.partition`: user provided mesh partitionition, defaults to metis if not specified
- `md.qmu.numberofpartitions`: number of partitions for semi-descrete qmu
- `md.qmu.variabledescriptors`
- `md.qmu.responsedescriptors`
- `md.qmu.method`: array of dakota method class
- `md.qmu.mass_flux_profile_directory`: directory for mass flux profiles
- `md.qmu.mass_flux_profiles`: list of mass flux profiles
- `md.qmu.mass_flux_segments`
- `md.qmu.adjacency`
- `md.qmu.vertex_weight`: weight applied to each mesh vertex

### 2.3.30 Results

One can display all these fields by typing:

```
>> md.results
```

### 2.3.31 Radar overlay parameters

One can display all these fields by typing:

```
>> md.radaroverlay
```

- `md.radaroverlay.pwr`: radar power image (matrix)
- `md.radaroverlay.x`: corresponding x coordinates
- `md.radaroverlay.y`: corresponding y coordinates

### 2.3.32 Miscellaneous

One can display all these fields by typing:

```
>> md.miscellaneous
```

- `md.miscellaneous.notes`: notes in a cell of strings
- `md.miscellaneous.name`: model name
- `md.miscellaneous.dummy`: empty field to store some data

# Chapter 3

## Mesh

### 3.1 Domain outline

To mesh the domain, one needs a file containing all the coordinates of the domain outline in an [Argus](#) format. These files have a `exp` extension. Here is an example of such a file for a square glacier:

```
## Name:DomainOutline
## Icon:0
# Points Count Value
5 1.000000
# X pos Y pos
0 0
1000000 0
1000000 1000000
0 1000000
0 0
```

This format is extensively used by ISSM. One can use `expmaster`, to generate and manage [Argus](#) files.

### 3.2 Trimesh Mesher

`Trimesh` is a wrapper of [triangle](#) developed by [Jonathan Shewchuk](#). It generates unstructured isotropic meshes:

```
>> md=setmesh(md,'DomainOutline.exp',5000);
```

The first argument is the model you are working on, the second argument is the file from ARGUS containing the Domain Outline, and the last argument is the density of the mesh (the mean distance between two nodes). To see how the mesh looks like, one can type:

```
>> plotmodel(md,'data','mesh');
```

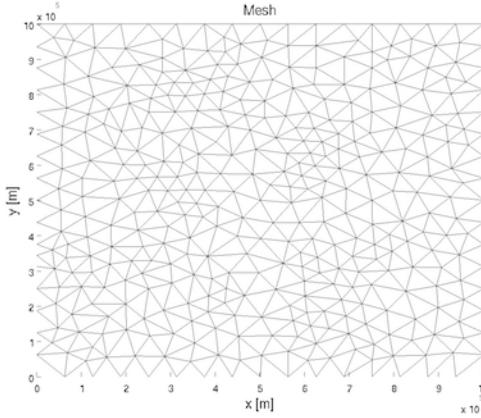


Figure 3.1: Mesh

### 3.3 Bamg Mesher

ISSM includes a mesh adaptation capability embeded in the code, inspired by [BAMG](#) developped by [Frederic Hecht](#), and [YAMS](#) developped by [Pascal Frey](#).

#### 3.3.1 Isotropic mesh generation

##### 3.3.1.1 Domain

To mesh the domain, you need a file containing all the coordinates of the domain outline in an Argus format. Assuming that this file is `DomainOutline.exp`

```
>> md=bamg(md,'DomainOutline.exp');
```

##### 3.3.1.2 hmin/hmax

The minimum and maximum edge lengths can be specified by `hmin` and `hmax` options:

```
>> md=bamg(md,'DomainOutline.exp','hmax',1000);
```

##### 3.3.1.3 hVertices

One can specified the edge length of domain outline vertices. `NaN` is used if not required.

```
>> h=[1000 100 100 100];
>> md=bamg(md,'DomainOutline.exp','hmax',1000,'hVertices',h);
```

#### 3.3.2 Anisotropic mesh generation

##### 3.3.2.1 field/err

The option `field` can be used with the option `err` to generate a mesh adapted to the field given in input for the error `err`:

```
>> md=bamg(md,'field',md.inversion.vel_obs,'err',1.5);
```

Several fields can be used:

```
>> md=bamg(md,'field',[md.inversion.vel_obs md.geometry.thickness],'err',[1.5 20]);
```

##### 3.3.2.2 gradation

The ratio of the lengths of two adjacent edges is controlled by the option `Gradation`:

```
>> md=bamg(md,'field',md.inversion.vel_obs,'err',1.5,'gradation',3);
```

### 3.3.2.3 anisomax

The factor of anisotropy (ratio between the lengths of two edges belonging to the same triangle) can be changed by option **aniso**. A factor of anisotropy equal to 1 will result in an isotropic mesh generation.

```
>> md=bamg(md,'field',md.vel_obs,'err',1.5,'anisomax',1);
```

# Chapter 4

## Model parameterization

### 4.1 Mask definition

The solver will use two different boundary conditions depending on the nature of the ice sheet system. An ice shelf will slide on the water whereas there is friction between an ice sheet and the bedrock for the grounded ice. The model must contain this field that tells whether the element is on an ice shelf or on an ice sheet. If the whole domain is an ice shelf, the following command may be used:

```
>> md=setmask(md,'all','');
```

and for an ice sheet:

```
>> md=setmask(md,'','');
```

If the geometry is more complex (marine ice sheet, ice shelf with ice rises), ARGUS files must be used. The following command will set the geometry:

```
>> md=setmask(md,'Iceshelves.exp','Islands.exp');
```

The first argument is the model and the two other arguments are the files containing the coordinates of the ice shelf included in the Domain Outline and the part of grounded ice included in the ice shelf part (Islands or ice rises).

### 4.2 Parameter file

To run a simulation, the solution sequence needs many parameters: physical constants, number of iterations, relaxation constant, thickness and surface of the glacier, etc. All this information must be specified in a file called *parameter file*, which is a matlab script. The majority of the parameters have default values that can be modified but other parameters (such as the geometry) must be specified in the *parameter file*. The parameterization must be done on a two dimensional mesh. The parameters will be automatically extruded if the mesh is extruded.

A model can be parameterize using the following command:

```
>> md=parameterize(md,'Parameters.par');
```

The first argument is the model, and the second argument between quotes is the file containing all the parameters' values.

#### 4.2.1 Interpolation routines

Several interpolation routines can be used in order to interpolate datasets onto the mesh vertices: The routine `ContourToMesh` is used to flag the nodes and/or elements that are within a contour from an Argus contour and a mesh. For example:

```
gridinsidefront=ContourToMesh(md.mesh.elements,md.mesh.x,md.mesh.y,expread('Front.exp',1),'node');
```

To interpolate a field from a structured grid to an unstructured mesh (or any list of points), one can use `InterpFromGridToMesh`:

```
data_mesh=InterpFromGridToMesh(x_grid,y_grid,data,x_mesh,y_mesh)
```

To interpolate a field from a 2d mesh to a 2d mesh (or any list of points), one can use `InterpFromMeshToMesh2d`:

```
data_mesh2=InterpFromMeshToMesh2d(index_mesh1,x_mesh1,y_mesh1,data,x_mesh2,y_mesh2)
```

To interpolate a field from a 3d mesh to a 3d mesh (or any list of points), one can use `InterpFromMeshToMesh3d`:

```
data_mesh2=InterpFromMeshToMesh3d(index_mesh1,x_mesh1,y_mesh1,z_mesh1,data,x_mesh2,y_mesh2,z_mesh2)
```

#### 4.2.2 Example

```
%%%%%%%%%%%%%%%
% GEOMETRY %%%%%%%%
disp('      reading thicknesses');
md.geometry.thickness=InterpFromFile(md.mesh.x,md.mesh.y,thicknesspath,10);

disp('      reading dem');
md.geometry.surface=InterpFromFile(md.mesh.x,md.mesh.y,surfacepath,10);

%get bed
md.geometry.bed=md.geometry.surface-md.geometry.thickness;

%%%%%%%%%%%%%%%
% OBSERVATIONS %%%%%%%%
disp('      reading velocities');
md=plugvelocities(md,velocitypath,0);

disp('      loading temperature');
md.initialization.temperature=InterpFromFile(md.mesh.x,md.mesh.y,temperaturepath,253);

disp('      creating mass balance rates');
md.surfaceforcings.mass_balance=InterpFromFile(md.x,md.y,massbalancepath,1);

disp('      loading geothermal flux');
load(heatfluxpath);
md.basalfocings.geothermalflux=InterpFromGrid(x_m,y_m,heatflux,md.mesh.x,md.mesh.y,80);

%%%%%%%%%%%%%%%
% MATERIAL %%%%%%%%
%flow law
disp('      creating flow law paramters');
md.materials.rheology_n=3*ones(md.mesh.numberofelements,1);
md.materials.rheology_B=paterson(md.initialization.temperature);

%%%%%%%%%%%%%%%
% BOUNDARY CONDITIONS %%%%%%%%
%drag md.drag or stress
md.friction.coefficient=300*ones(md.mesh.numberofvertices,1); %q=1.

%Take care of iceshelves: no drag md.drag
pos=find(md.mask.elementonfloatingice);
md.friction.coefficient(md.mesh.elements(pos,:))=0;
md.friction.p=ones(md.mesh.numberofelements,1);
md.friction.q=ones(md.mesh.numberofelements,1);
```

---

```
%Create ice front
md=SetMarineIceSheetBC(md);
```

### 4.3 Verbosity levels

The verbosity level is set by the field `md.verbose`. This field is a matlab object, which activates and deactivates different verbosity levels. By default, all verbosity levels are deactivated. For a complete list of available levels:

```
>> help verbose
VERBOSE class definition

Available verbosity levels:
mprocessor : model processing
module     : modules
solution    : solution sequence
solver      : solver info (extensive)
convergence : convergence criteria
control     : control method
qmu        : sensitivity analysis
```

To activate the levels `module` and `solution`:

```
>> md.verbose=verbose();
>> md.verbose=verbose('module',true,'solution',true);
```

### 4.4 Model extrusion (optional)

One can extrude the mesh, in order to use a 3 dimensional model (Pattyn's higher order model and Full Stokes model). This step is not mandatory if the user wants to keep a 2d model, skip this section. To extrude the mesh, type the following command:

```
>> md=extrude(md,8,3);
```

The first argument is the model as usual. The second argument is the number of horizontal layers. A high number of layers gives a better precision for the simulations but creates more elements which require a longer computational time. Usually a number between 7 and 10 is a good balance. The third argument is called the extrusion exponent. Interesting things are happening near the bedrock usually and users might want to refine more the lower layers than the upper ones. An extrusion exponent of 1 will create a mesh with layers vertically equally distributed. The higher the extrusion exponent, the more refined the base. An extrusion exponent of 3 or 4 is generally enough.

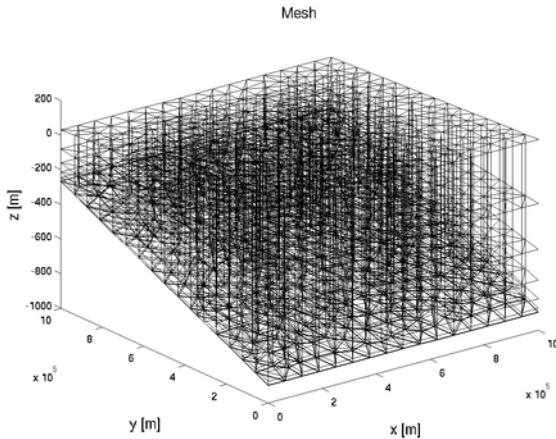


Figure 4.1: Extruded mesh

## 4.5 Ice flow approximations

ISSM has the capability to compute the flow of a glacier with 4 different models:

- Hutter's ice sheet model (2d and 3d)
- MacAyeal/Morland's shelfy stream model (2d and 3d)
- Blatter/Pattyn's higher order model (3d: extruded mesh only)
- Full Stokes' model (3d: extruded mesh only)

The ice flow model is specified for each element of the mesh. To assign the models to the elements, as an example the following command can be used:

```
>> md=setflowequation(md,'pattyn','Pattyn.exp','macayeal',md.mask.elementonfloatingice,'fill','hutter')
```

The routine `setflowequation` works like `plotmodel`: it requires an even number of argument (without counting `md` itself). There are five possible options:

- 'hutter'
- 'macayeal'
- 'pattyn'
- 'stokes'
- 'fill'

The first four options must be followed by one of the following argument:

- An ARGUS file containing a closed contour, the elements inside the contour will be assigned to the model given by the option. If user wants to assign the model to the elements outside the domain, add '' to the name of the domain file (ex: 'Pattyn.exp')
- A vector of size `md.numberofelements` holding 0, and 1 on the elements that the user had flagged. The model given by the option will be assigned to the elements flagged only.
- 'all' if the user wants to assign the model to all the elements

The last option 'fill' must be followed by the name of the model that the user wants the other elements (that have not been flagged by the other option) assigned to. All options are not required to be used. The previous example assigns the model of Pattyn for the element inside the contour `Pattyn.exp`, the model of MacAyeal for the elements located on the ice shelf. The other elements are Hutter's elements. If the user wants to use MacAyeal's model only, type the following command:

```
>> md=setflowequation(md,'macayeal','all');
```

# Chapter 5

## Solutions

### 5.1 Launching a solution sequence

To run a simulation, use the following command:

```
>> md=solve(md,ThermalSolutionEnum);
```

The first argument is the model, the second is the nature of the simulation one wants to run. The supported solutions are:

- `DiagnosticSolutionEnum` - steady state velocity computation.
- `SteadystateSolutionEnum` - steady state velocity and temperature computation (3d only).
- `PrognosticSolutionEnum` - evolution of the thickness after one time step.
- `ThermalSolutionEnum` - temperature field in steady state or transient (depending on dt).
- `TransientSolutionEnum` - evolution of an ice sheet system (velocity, geometry, temperature, ... ).

### 5.2 Parallel computing

ISSM can be run in parallel on clusters or on multi-core computers. This subsection shows how to use this capability.

#### 5.2.1 Setting up the environment to use the parallel mode

We assume users have correctly setup ISSM. Every cluster is different and one might need to create a new cluster file in `ISSM_TIER/src/m/classes/clusters/`. In most cases, the `generic` cluster can be used:

```
>> md.cluster=generic('name',machine_name)
```

For a local machine, the command `oshostname()` can be used:

```
>> md.cluster=generic('name',oshostname())
```

Many parameters, such as the number of processors, are fields of `md.cluster`. Once those parameters are setup, the solution sequences are called the same way:

```
md=solve(md,DiagnosticSolutionEnum);
```

#### 5.2.2 password-less SSH login

In order to facilitate use of clusters that might be protected by passwords, or to avoid having to input password for each run, one can either set-up a public key authentication or a tunnel between the local host and the cluster.

### 5.2.2.1 SSH passthrough

You need to have a SSH public/private key pair. If you do not, you can create a SSH public/private key pair by typing the following command and following the prompts (no passphrase necessary):

```
$your_localhost% ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/username/.ssh/id_rsa):RETURN
Enter passphrase (empty for no passphrase):RETURN
Enter same passphrase again:RETURN
Your identification has been saved in /Users/username/.ssh/id_rsa.
Your public key has been saved in /Users/username/.ssh/id_rsa.pub.
```

Two files were created: your private key `/Users/username/.ssh/id_rsa`, and the public key `/Users/username/.ssh/id_rsa.pub`. The private key is read-only and only for you, it is used to decrypt all correspondence encrypted with the public key. The content of the public key Its content is then copied in file `~/.ssh/authorized_keys` of the system you wish to SSH to without being prompted for a password.

```
$your_localhost%scp ~/.ssh/id_rsa.pub username@your_remotesthost:~
```

Now on your remote host, copy the content of `id_rsa.pub`:

```
$your_remotesthost%cat ~/id_rsa.pub >> ~/.ssh/authorized_keys
$your_remotesthost%rm ~/id_rsa.pub
```

### 5.2.2.2 Tunneling

Another possibility is to establish an SSH tunnel between the local host and the cluster. Open a shell, and connect to the cluster using ssh, by typing:

```
$ ssh -L 1025:localhost:22 login@cluster
```

The port that will be tunneled is the port 1025. To be able to use the tunnel, you will have to change the port setting in the `md.cluster.port` from 0 to 1025. Once this is done, solutions can be solved the exact same way.

### 5.2.3 FAQ

The following message appears in the errlog file when launching my job in parallel:

```
mpdrun_wilkes.jpl.nasa.gov: cannot connect to local mpd (/tmp/mpd2.console_name);
possible causes:
1. no mpd is running on this host
2. an mpd is running but was started without a "console" (-n option)
~
```

This message means that the MPI (Message Passing Interface) server, called mpd, is not running. Therefore, no parallel jobs can run on the cluster. To solve this issue, just type, at the command prompt on the server side (if for example your cluster has 8 cpus):

```
mpd --ncpus=8 &
```

This will launch the MPI server to manage 8 cpus on the cluster.

# Chapter 6

## Advanced features

### 6.1 Inverse methods

Inverse methods have been implemented in ISSM for the diagnostic solution and for the balance thickness solution. In the diagnostic solution, 2 parameters can be inverted: the basal drag parameter,  $k$ , in:

$$\tau_b = -k^2 N^r \|\mathbf{v}\|^{s-1} \mathbf{v}_b \quad (6.1)$$

and the depth-averaged ice hardness,  $B$ , in Glen's flow law:

$$\mu = \frac{B}{2 \left( \dot{\varepsilon}_e^{1-\frac{1}{n}} \right)} \quad (6.2)$$

For the balance thickness solution, 3 parameters can be optimized: the depth-averaged velocity components, and the apparent mass balance.

This section explains how to launch an inverse method and how optimization parameters must be tuned. An inversion can be scheduled by setting the `md.inversion.iscontrol` flag to 1 as follows (0 if not).

```
>> md.inversion.iscontrol=1;
```

All the parameters used in a control method can be displayed by typing

```
>> md.inversion
```

#### 6.1.1 Objective functions

Inversions can use different types of objective functions. The objective functions must be entered in `md.inversion.cost_functions` for each optimization step:

- 101: Surface Absolute Velocity Misfit
- 102: Surface Relative Velocity Misfit
- 103: Surface Logarithmic Velocity Misfit
- 104: Surface Logarithmic Velocity compoments Misfit
- 105: Surface Average Velocity Misfit
- 201: Thickness Absolute Misfit
- 501: Drag Coefficient Gradient square
- 502: B Gradient square
- 503: Thickness Gradient square

### 6.1.1.1 Absolute misfit

This is the classic way of calculating a misfit between a modeled and observed velocity field:

$$\mathcal{J}(\mathbf{v}) = \int_S \frac{1}{2} \left( (v_x - v_x^{\text{obs}})^2 + (v_y - v_y^{\text{obs}})^2 \right) dS \quad (6.3)$$

Where:

- $v_x$  is the x component of the glacier modeled velocity
- $v_y$  is the y component of the glacier modeled velocity
- $v_x^{\text{obs}}$  is the x component of the glacier observed velocity
- $v_y^{\text{obs}}$  is the y component of the glacier observed velocity

### 6.1.1.2 Relative misfit

The relative misfit is defined as follows:

$$\mathcal{J}(\mathbf{v}) = \int_S \frac{1}{2} \left( \frac{(v_x - v_x^{\text{obs}})^2}{(v_x^{\text{obs}} + \varepsilon)^2} + \frac{(v_y - v_y^{\text{obs}})^2}{(v_y^{\text{obs}} + \varepsilon)^2} \right) dS \quad (6.4)$$

Where:

- $v_x$  is the x component of the glacier modeled velocity
- $v_y$  is the y component of the glacier modeled velocity
- $v_x^{\text{obs}}$  is the x component of the glacier observed velocity
- $v_y^{\text{obs}}$  is the y component of the glacier observed velocity
- $\varepsilon$  is a minimum velocity used to avoid the observed velocity being equal to zero. One takes usually `md.inversioneps=eps;` (Matlab's epsilon)

### 6.1.1.3 Logarithmic misfit

$$\mathcal{J}(\mathbf{v}) = \int_S \left( \log \left( \frac{\|\mathbf{v}\| + \varepsilon}{\|\mathbf{v}^{\text{obs}}\| + \varepsilon} \right) \right)^2 dS \quad (6.5)$$

Where:

- $\mathbf{v}$  is the glacier modeled velocity magnitude
- $\mathbf{v}^{\text{obs}}$  is the glacier observed velocity magnitude
- $\varepsilon$  is a minimum velocity used to avoid the observed velocity being equal to zero.

### 6.1.1.4 Thickness misfit

$$\mathcal{J}(H) = \int_{\Omega} \frac{1}{2} (H - H^{\text{obs}})^2 d\Omega \quad (6.6)$$

Where:

- $H$  is the ice thickness
- $H^{\text{obs}}$  is the measured ice thickness

### 6.1.1.5 Drag gradient

$$\mathcal{J}(k) = \int_B \gamma \frac{1}{2} \|\nabla k\|^2 dB \quad (6.7)$$

Where:

- $\gamma$  is a Tikhonov regularization parameter

### 6.1.1.6 Thickness gradient

$$\mathcal{J}(k) = \int_{\Omega} \gamma \frac{1}{2} \|\nabla H\|^2 d\Omega \quad (6.8)$$

Where:

- $\gamma$  is a Tikhonov regularization parameter

## 6.1.2 Inversion parameters

### 6.1.2.1 Control parameters

One can choose which parameter to optimize. This is registered in `md.inversion.control_parameters` as a string ('MaterialsRheologyBbar' or 'FrictionCoefficient').

```
>> md.inversion.control_parameters='FrictionCoefficient';
```

### 6.1.2.2 Constraints

Usually, the viscosity or the drag are physically constrained. For example the viscosity can not be negative. To prevent the inversed parameters from being out of the physical range, enter the constraints values in `md.inversion.min_parameters` and `md.inversion.max_parameters`. If one does not wish to constrain the inverse parameters, enter NaN.

```
>> md.inversion.min_parameters=5;
>> md.inversion.max_parameters=300;
```

### 6.1.2.3 Optimization parameters

- `nsteps`

The number of optimization iterations is set in `nsteps`.

```
>> md.inversion.nsteps=25;
```

Warning : the other control parameters (`md.inversion.gradient_scaling`, `md.inversion.step_threshold`, `md.cm_responses` and `md.maxiter`) must have a length equal to `md.inversion.nsteps`.

- `gradient_scaling`

At each iteration, the routine evaluates the gradients of the misfit with respect to the inversed parameters spatial distribution and updates each parameter spatial distribution using a scalar as follows:

$$\alpha \in [0, \text{gradient\_scaling}] \quad p^{\text{new}} = p^{\text{old}} - \alpha \nabla_p \mathcal{J} \quad (6.9)$$

`optscal` is the scaling factor on gradient direction during optimization. We recommend to use  $10^8$  for the viscosity and 20 for the drag:

```
>> md.inversion.gradient_scaling=20*ones(md.maxiter.nsteps,1);
```

- `cost_functions`

If one wants half logarithmic and half absolute, use the following command:

```
>> md.inversion.cost_functions=...
    [103*ones(floor(md.inversion.nsteps/2),1); 101*ones(ceil(md.inversion.nsteps/2),1)];
```

- `maxiter_per_step`

At each iteration, the routine search for a scalar between 0 and 1. It stops when it reaches a maximum number of iteration `md.inversion.maxiter_per_step`. We recommend :

```
>> md.inversion.maxiter_per_step=20*ones(md.inversion.nsteps,1);
```

### 6.1.3 Launching a control method

To launch a control method, two solutions can be called. Either '`diagnostic`' where only the velocity is computed, or '`steadystate`'. With the latter, the thermo-dynamic equilibrium will be used. To launch a control method, type:

```
>> md=solve(md,SteadystateSolutionEnum);
```

## 6.2 Rifts

ISSM allows simulation of rifts. This section explains how to create a model that includes rifts, and how to control their behaviour.

### 6.2.1 Rifts creation

Rifts can be included right between the phase where the mesh is created, and the phase where the geography is setup. Rifts that should be included in the model must be present in an Argus type file. Each rift should be represented by an open loop set of points. Infinite numbers of rifts can be included, provided they do not intersect with the domain outline, or any other rift. This point is particularly important as there are no checks on intersections at the meshing phase. For example, a file including two straight rifts could look like, `Rifts.exp`:

```
## Name:Rift1
## Icon:0
# Points Count  Value
2 1.000000
# X pos Y pos
0 0
50000 0

## Name:Rift2
## Icon:0
# Points Count  Value
2 1.000000
# X pos Y pos
0 10000
50000 10000
```

this file includes two horizontal rifts of 50 km long, separated by 10 km. In order to create a model with these rifts, one would do:

```
>> md=model;
>> md=setmesh(md,'DomainOutline.exp','Rifts.exp',4000);
>> md=meshprocessrifts(md);
>> md=setmask(md,'Iceshelves.exp','Islands.exp');
>> etc ...
```

The rest of the process is similar. This will create a rifts structure in the model `md`. The rifts structure holds as many members as there are rifts in `Rifts.exp`. The key fields in the rifts structure, are the fill and friction. Fill can be either 1 (for water), 2 (for air) and 3 (for ice). Fill determines the pressure on each flank of the rifts that is being applied. friction is a coefficient between the shear stress exerted on the rift flanks, and the differential tangential velocity between both flanks.

### 6.2.2 Rifts tip refining

Rifts in a mesh will not modify the type of meshing occurring during the mesh phase. To impact the mesh, one can use the `riftstiprefine.m` routine. This routine will ensure that the rift tips are correctly refined, to take into account the tip stress singularity. Use of this routine is as follows:

```
>> md=model;
>> md=setmesh(md,'DomainOutline.exp','Rifts.exp',4001);
>> md=rifftipsrefine(md,2000,30000);
>> md=meshprocessrifts(md);
>> md=setmask(md,'Iceshelves.exp','Islands.exp');
>> etc ...
```

the first argument is the model, the second argument the tip area resolution, and the third is the size of the circle around the tips where mesh refinement should occur.

### 6.2.3 Rifts in parameter file

The structure rifts can be modified in any parameter file. We do not advise touching anything except the fill and friction for each one of the rifts in the structure. For example, inclusion of the following lines in the parameter file should be enough:

```
>> for i=1:md.numrifts,
>>     md.rifts.riftstruct(i).fill=WaterEnum() %include water in the rifts
>>     md.rifts.riftstruct(i).friction=10^11 %friction parameter sigma=10^11*dv_t
>> end
```

Of course, different frictions and fill could be applied, according to the physics being captured.

### 6.2.4 Solving for rifts

Rifts are only allowed when using MacAyeal type elements, in 2d meshes. For now, 3d meshes are not supported. Nothing is needed to take rifts into account in the solve phase. A simple:

```
>> md=solve(md,DiagnosticSolutionEnum);
```

will suffice. Bear in mind, rifts are handled using penalty methods, to ensure that penetration of rift flanks does not occur. This can be very computationally expensive, as penalty methods tend to lead to zigzagging of contact. A stable set of constraints strategy has been implemented, which should guarantee convergence, but which can be slow. Users should also try to minimize zigzagging by refining the mesh where needed. In case zigzagging becomes too intense, locking of the zigzagging penalties will occur, which ensures convergence, but which can lead to bad results in a physical sense. Detecting penalty locking should give users an idea on where to refine the mesh.

### 6.2.5 Rifts plotting

Rifts can be plotted using the following special plots:

```
>> plotmodel(md,'data','rifts','data','riftpenetration','data','riftvel','data','riftrelvel');
```

these three plots will give users a view of which parts of the rifts are opening, closing, at which relative speed, etc ...

### 6.2.6 Rifts when using Yams mesh adaptation

Rifts can be used in conjunction with the Yams mesh adaptation routine, by adding the Rifts.exp file defining rifts contours to the 'riftoutline' option of meshyams. For ex:

```
>> md=meshyams(md,'domainoutline','DomainOutline.exp','riftoutline','Rifts.exp','velocities','vel.mat')
```

### 6.2.7 Adding rifts to an existing mesh

In case users want to use an existing mesh, rifts can still be added on. The format for the rifts file is in this case slightly different:

```

## Name:ContourAroundRift1
## Icon:0
# Points Count  Value
5 1
# X pos Y pos
-100 -100
50100 -100
50100 +100
-100 +100
-100 -100

## Name:Rift1
## Icon:0
# Points Count  Value
2 500
# X pos Y pos
0 0
50000 0

## Name:ContourAroundRift2
## Icon:0
# Points Count  Value
5 1
# X pos Y pos
-100 900
50100 900
50100 1100
-100 1100
-100 900

## Name:Rift2
## Icon:0
# Points Count  Value
2 1000
# X pos Y pos
0 10000
50000 10000

```

The format is made of pairs of rift contours with the corresponding rift profile. The rift contour is a closed contour that envelopes the rift. The rift that follows needs to be completely included in it. The rift density (here, 500 and 1000 respectively) is very important, as it will decide the density of the mesh around the rift. Do not specify 1, as this will try to include a rift in the mesh with a 1 m mesh density, which will probably result in a memory exhaustion problem for the local machine running ISSM.

### 6.3 Quantifications of Margins and Uncertainties with Dakota

Given a response that is a function of multiple variables in a local reliability analysis [e.g., Coleman and Steele, Experimentation and Uncertainty Analysis for Engineers]:

$$r = r(x_1, x_2, \dots, x_n) \quad (6.10)$$

the sensitivities are defined as:

$$\theta_i = \frac{\delta r}{\delta x_i} \quad (6.11)$$

and if each of the variables is independent, the error propagation equation gives the variance:

$$\sigma_r^2 = \sum_{i=1}^n \theta_i^2 \sigma_i^2 \quad (6.12)$$

The propagated variance consists of two factors for each input variable:

- Sensitivity: computed from the function evaluations using finite differences. Dependent on the specified step size. To vary, must re-run the function evaluations.
- Standard deviation: specified for the variable distribution. Used directly from the user input. To vary, may use a spreadsheet.

Dividing the error propagation equation by  $\sigma_r^2$  gives the importance factors for each variable. The mean of the response is taken to be the response at the nominal values of the variables.

# Chapter 7

## Plotting

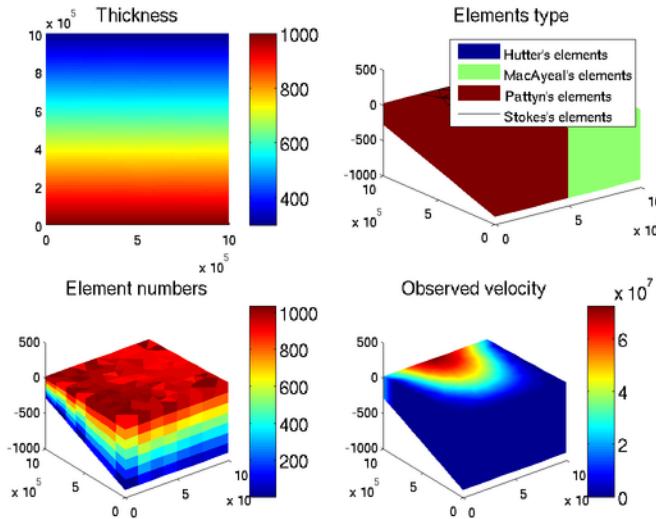
### 7.1 Standard plots

`plotmodel` takes the model `md` as first argument and then an even number of options (as in the function `setelementstype`, or `solve`). To plot a given field, use the option '`data`' followed by the field one wants to plot. For the thickness:

```
>> plotmodel(md,'data',md.geometry.thickness)
```

You can plot several fields at the same time but you have to add the argument '`data`' before each field you want to plot:

```
>>  
plotmodel(md,'data',md.geometry.thickness,'data','mesh','data',[1:md.mesh.numberofelements])
```

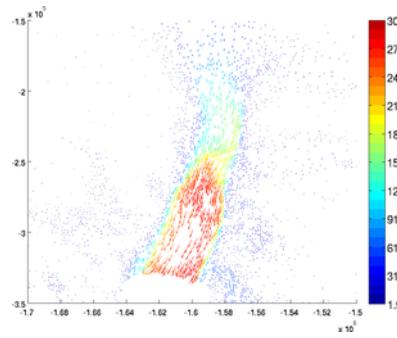


This can work for any field of length `md.mesh.numberofelements` or `md.mesh.numberofvertices`.

### 7.2 Quiver plots

For 2d or 3d fields, a generic color plot cannot be used (except component by component). The '`data`' used by the function `plotmodel` must be a matrix of 2 or 3 columns. For example:

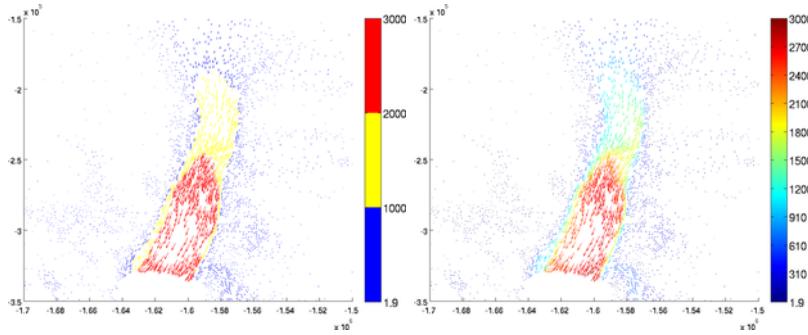
```
>> plotmodel(md,'data',[md.vx md.vy])
```



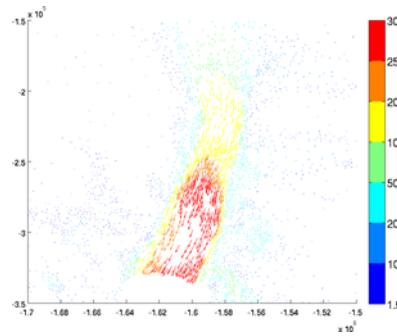
### 7.2.1 ColorLevels

The number of colors can be chosen by using the 'colorlevels' options. The user can specify a number of levels or a cell containing the values of color changes (See examples below).

```
>> plotmodel(md,'data',[md.vx md.vy],'colorlevels',3)
>> plotmodel(md,'data',[md.vx md.vy],'colorlevels',100)
```



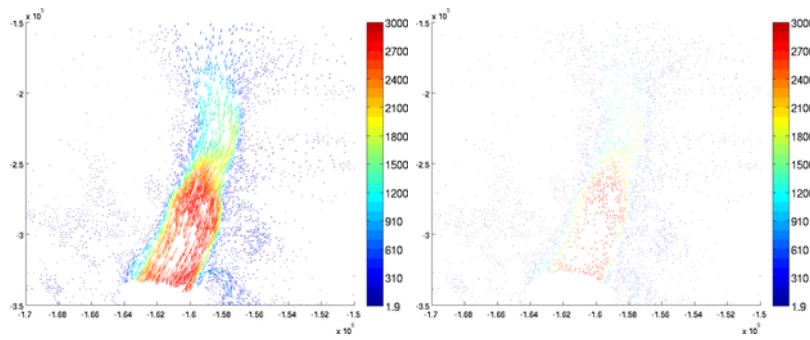
```
>> plotmodel(md,'data',[md.vx md.vy],'colorlevels',{100,200,500,1000,2000,2500})
```



### 7.2.2 Scaling

The arrows length can be modified with the 'scaling' options. The default value is 0.4. A higher scaling value will result in longer arrows.

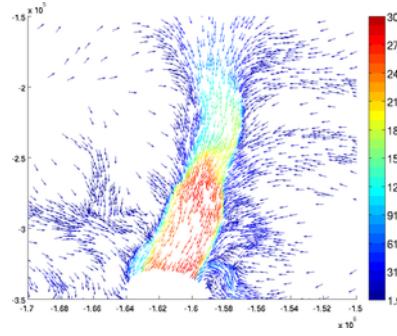
```
>> plotmodel(md,'data',[md.vx md.vy],'scaling',1)
>> plotmodel(md,'data',[md.vx md.vy],'scaling',0.1)
```



### 7.2.3 Autoscale

If the user wants all the arrows to have the same length, use the option '`autoscale`' set as '`off`'.

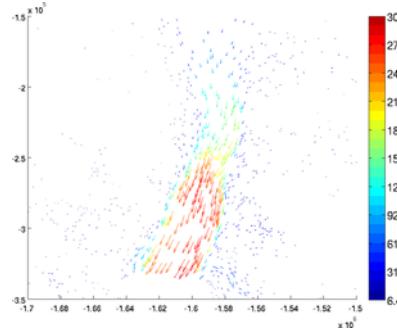
```
>> plotmodel(md,'data',[md.vx md.vy], 'autoscale','off')
```



### 7.2.4 Density

The number of arrows can be reduced with the option '`density`'. If the density is set as 3, only one arrow out of 3 will be displayed. This option is very useful when the mesh is very refined.

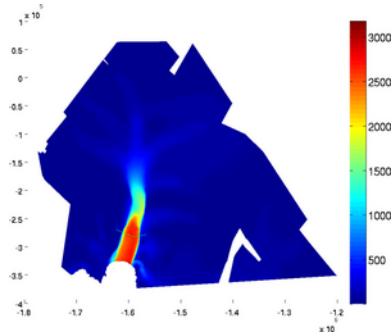
```
>> plotmodel(md,'data',[md.vx md.vy], 'density',3)
```



## 7.3 Section plots

The section plot can be used to display the value of a field on a given track. The option '`sectionvalue`' must be followed by the name of an Argus file which contained the coordinates of the points describing the profile (this file can be generated by `expmaster.m`). The resulting plot will be a curve in 2d and a colored surface in 3d (See example below).

```
>> plotmodel(md,'data',md.vel,'expdisp','track.exp')
```



```
>> plotmodel(md,'data',md.vel,'sectionvalue','track.exp')
```

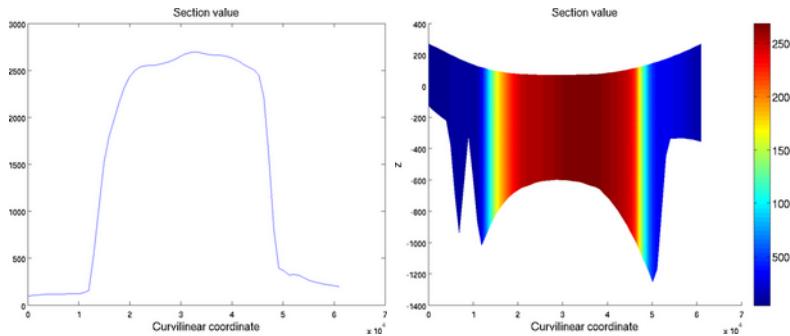


Figure 7.1: Section plot for 2D (left) and 3d (right) models

### 7.3.1 Section plots options

#### 7.3.1.1 Resolution

The horizontal and vertical (in 3d) resolution can be specified by the '`resolution`' option. It must be a list with the horizontal resolution followed by the vertical resolution (in meters). When not specified, the default resolution is displayed.

```
>> plotmodel(md,'data',md.vel,'sectionvalue','track.exp','resolution',[2*10^4 0])
```

```
>> plotmodel(md,'data',md.vel,'sectionvalue','track.exp','resolution',[10^3 0])
```

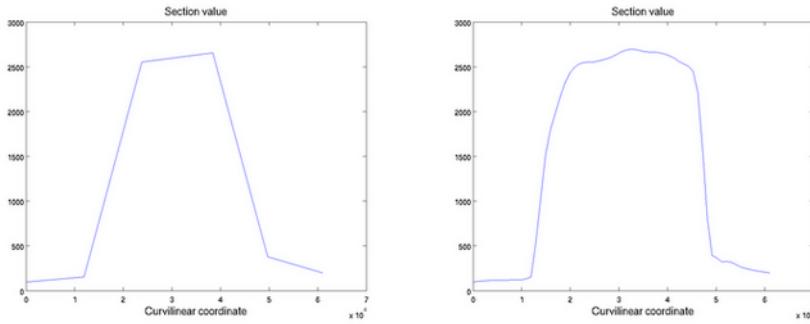
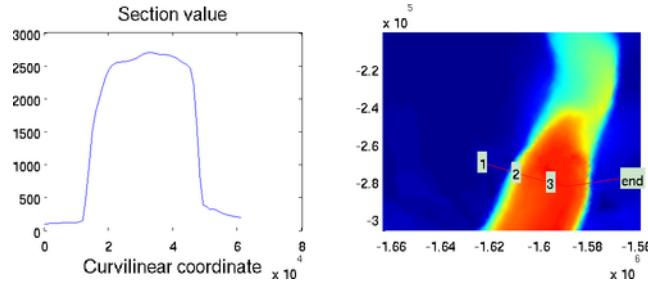


Figure 7.2: Horizontal resolution of  $2 \times 10^4$  m (left) and  $10^3$  m (right)

### 7.3.1.2 Show section

The profile used to create the section plot can be also plotted with the 'showsection' option.

```
>> plotmodel(md,'data',md.vel,'showsection','on')
```



## 7.4 Special plots

### 7.4.1 basaldrag

The special plot 'basal\_drag' displays the norm of the basal drag friction in kPa following formula:

$$\tau_b = -k^2 N^r \|v\|^{s-1} v_b \quad (7.1)$$

The x and y components of the basal drag can be displayed with the 'basal\_dragx' or 'basal\_dragy' special plots:

```
>> plotmodel(md,'data','basal_drag')
>> plotmodel(md,'data','basal_dragx')
```

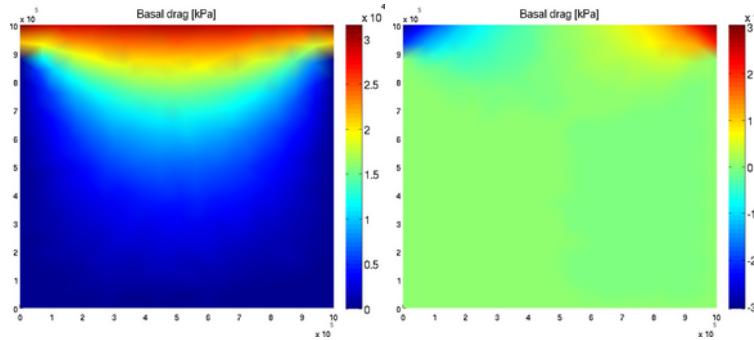
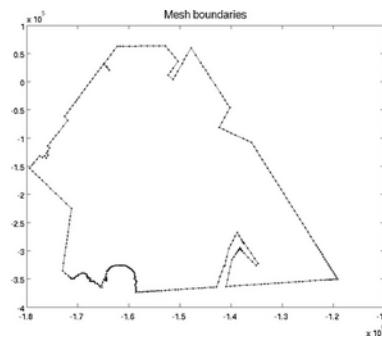


Figure 7.3: Basal friction norm (left) and Basal friction x-component (right)

### 7.4.2 boundaries

The special plot 'boundaries' displays the 2d boundaries of the domain (domain outline).

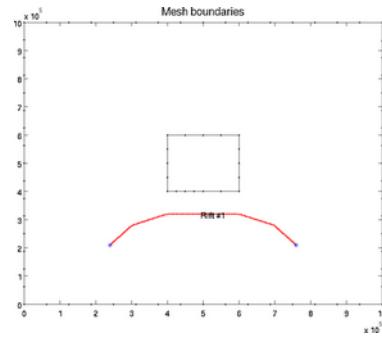
```
>> plotmodel(md,'data','boundaries')
```



### 7.4.3 boundariesrift

The special plot 'boundariesrift' displays the rifts included in the domain

```
>> plotmodel(md,'data','boundariesrift')
```

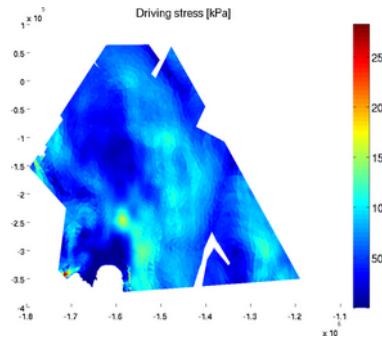


### 7.4.4 driving\_stress

The special plot 'driving\_stress' displays the basal drag friction in kPa following formula:

$$d = \rho g H \nabla s \quad (7.2)$$

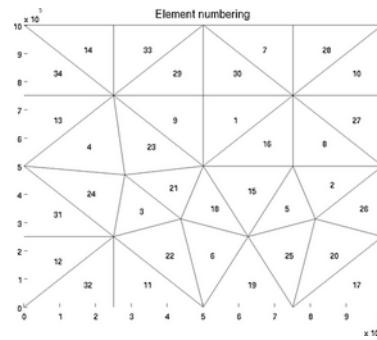
```
>> plotmodel(md,'data','driving_stress')
```



### 7.4.5 elementnumbering

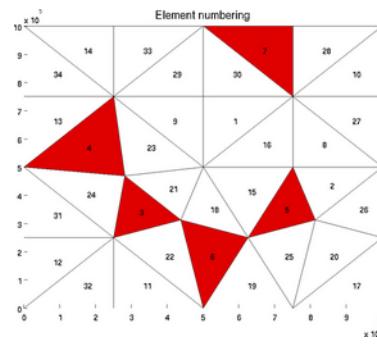
In the debugging process, it is often very useful to display all the elements next to their numbers. This is what the special plot 'elementnumbering' does:

```
>> plotmodel(md,'data','elementnumbering')
```



A given list of elements can be highlighted with the 'highlight' option:

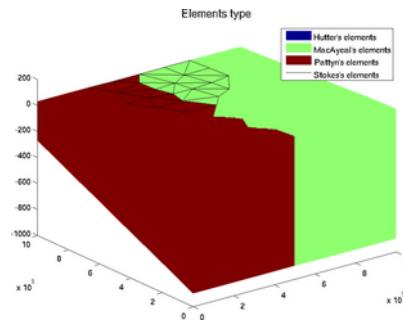
```
>> plotmodel(md,'data','elementnumbering','highlight',[3 4 5 6 7])
```



#### 7.4.6 elements\_type

The special plot 'elements\_type' displays the elements with a specific color for each formulation.

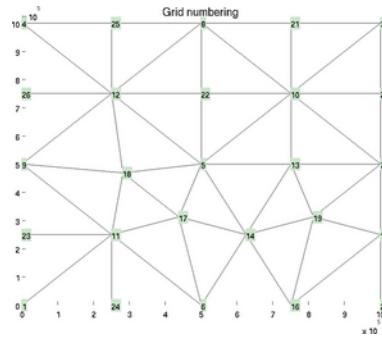
```
>> plotmodel(md,'data','elements_type')
```



#### 7.4.7 gridnumbering

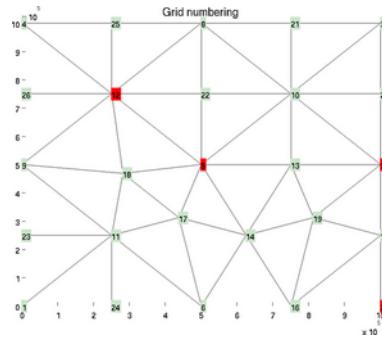
In the debugging process, it is often very useful to display all the grids next to their numbers. This is what the special plot 'gridnumbering' does:

```
>> plotmodel(md,'data','gridnumbering')
```



A given list of grids can be highlighted with the 'highlight' option:

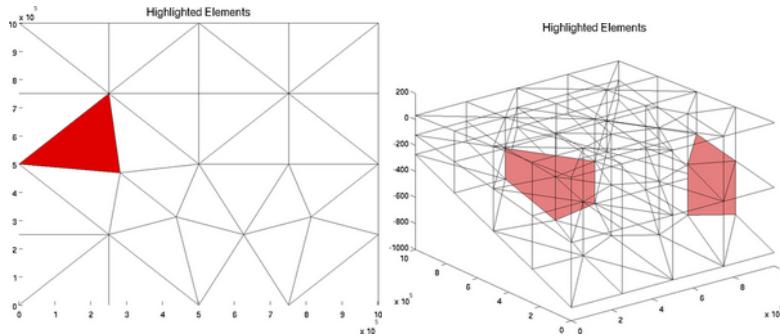
```
>> plotmodel(md,'data','gridnumbering','highlight',[2 5 7 12])
```



#### 7.4.8 highlightelements

The special plot 'highlightelements' is very similar to the plot 'elementnumbering'. It is another possibility to highlight one or several grids, but without indicating the number of all the elements. It is way faster for large models.

```
>> plotmodel(md,'data','highlightelements','highlight',5)
>> plotmodel(md,'data','highlightelements','highlight',[5 12])
```

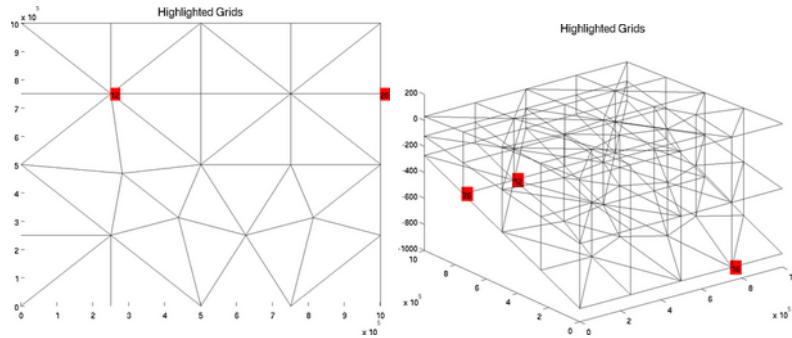


#### 7.4.9 highlightgrids

The special plot 'highlightgrids' is very similar to 'gridnumbering'. It is another possibility to highlight grids without indicating all the grids numbers. It is way faster for big models.

```
>> plotmodel(md,'data','highlightgrids','highlight',[12 20])
```

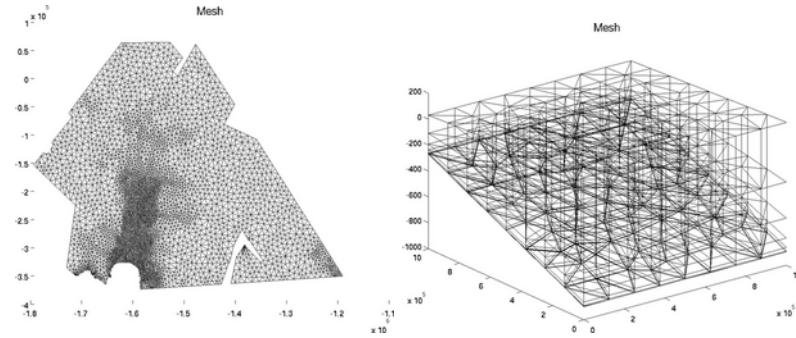
```
>> plotmodel(md,'data','highlightgrids','highlight',[12 16 26])
```



#### 7.4.10 mesh

The special plot 'mesh' displays the mesh of 2d or 3d model.

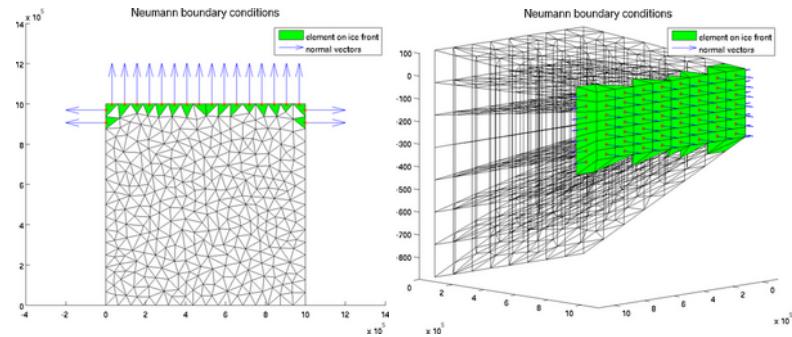
```
>> plotmodel(md,'data','mesh')
```



#### 7.4.11 pressureload

The special plot 'pressureload' displays the neumann boundary conditions, ie all the segments on ice front and the normal to these segments, for a 2d or 3d mesh.

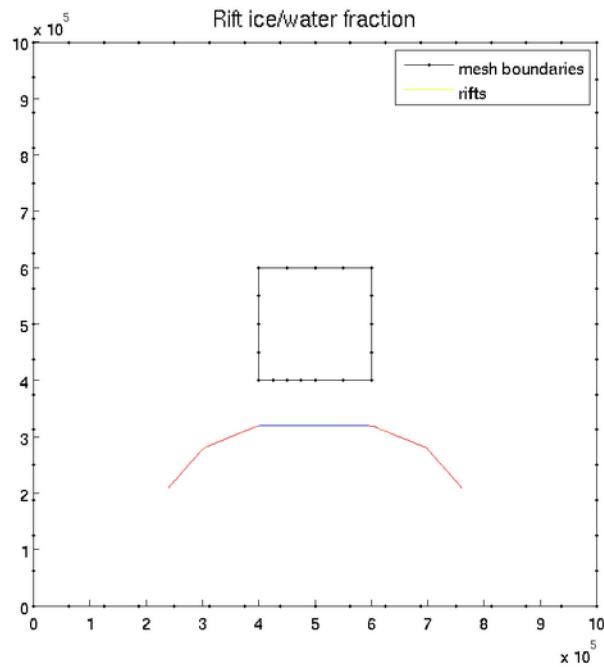
```
>> plotmodel(md,'data','pressureload')
```



#### 7.4.12 riftfraction

The special plot 'riftfraction' indicates the fraction of ice/water in a rift.

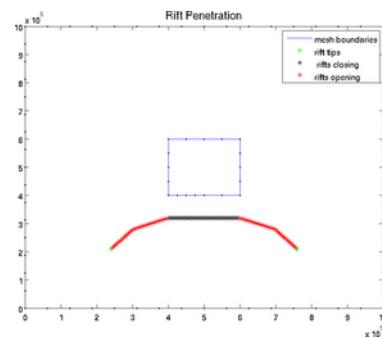
```
>> plotmodel(md,'data','riftfraction')
```



#### 7.4.13 riftpenetration

The special plot 'riftpenetration' indicates the parts of a rift opening and the one closing.

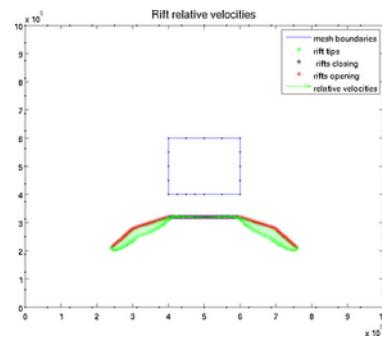
```
>> plotmodel(md,'data','riftpenetration')
```



#### 7.4.14 riftrelvel

The special plot 'riftrelvel' displays the relative velocity between the two edges of existing rifts.

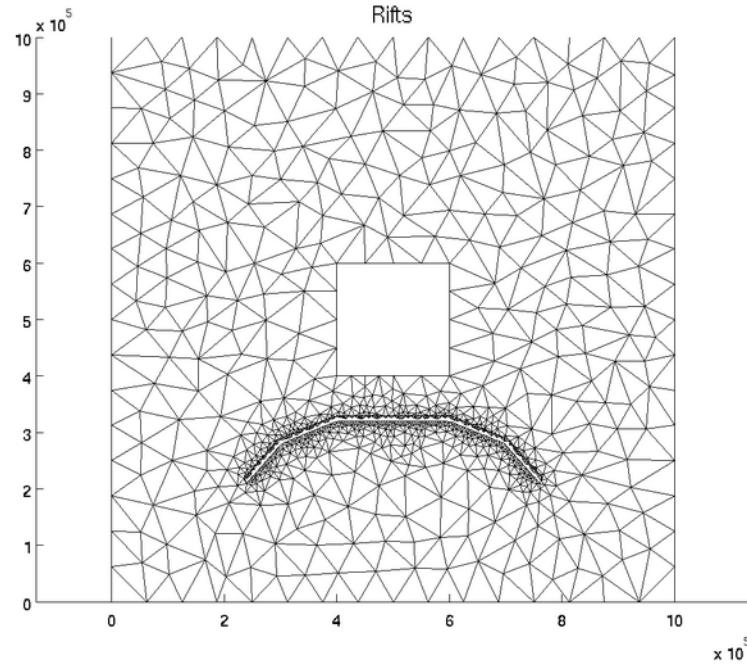
```
>> plotmodel(md,'data','riftrelvel')
```



#### 7.4.15 rifts

The special plot 'rifts' plot the mesh with an offset so that the rifts are visible. You can specify the offset using the option 'offset'.

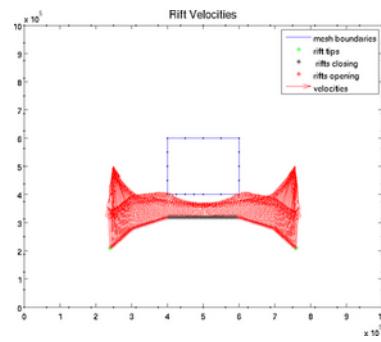
```
>> plotmodel(md,'data','rifts','offset',1000)
```



#### 7.4.16 riftvel

The special plot 'riftvel' displays the velocity along existing rifts.

```
>> plotmodel(md,'data','riftvel')
```



## 7.5 Plot options

Options in `plotmodel` comes as pairs: the option name must be followed by its value. For example, if one wants to remove the color bar, the option name is '`colorbar`' and the value 0:

```
>> plotmodel(md,'data',md.initialization.vel,'colorbar',0)
```

any options (except '`data`') can be followed by '#i' where i is the subplot number, or '`#all`' if applied to all plots. Example:

```
>> plotmodel(md,'data',md.initialization.vel,'data','mesh','view#2',3,'colorbar#all','on','axis#1','o')
```

### 7.5.1 Axis

#### 7.5.1.1 axis

Same as standard `axis` matlab option

```
>> plotmodel(md,'data',md.vel,'axis','tight')
```

#### 7.5.1.2 antzoom

The option '`antzoom`' is used for Antarctica models. When the user wants to zoom on a given region (Pine Island Glacier, Larsen,...)

```
>> plotmodel(md,'data',md.vel,'antzoom','ameryiceshelf')
```

For a complete list of the available basins, type `antzoom` in Matlab prompt.

#### 7.5.1.3 view

Same as standard `view` matlab option

```
>> plotmodel(md,'data',md.vel,'view',2)
```

#### 7.5.1.4 xlim

Same as standard `xlim` matlab option

```
>> plotmodel(md,'data',md.vel,'xlim',[10^5 2*10^5])
```

#### 7.5.1.5 ylim

Same as standard `ylim` matlab option

```
>> plotmodel(md,'data',md.vel,'ylim',[10^5 2*10^5])
```

### 7.5.1.6 zlim

Same as standard `zlim` matlab option

```
>> plotmodel(md,'data',md.vel,'zlim',[10^5 2*10^5])
```

## 7.5.2 Title and labels

### 7.5.2.1 title

Same as standard `title` matlab option

```
>> plotmodel(md,'data',md.vel,'title','Ice velocity [m/yr]')
```

### 7.5.2.2 fontsize

Same as standard `fontsize` matlab option

```
>> plotmodel(md,'data',md.vel,'title','Ice velocity [m/yr]', 'fontsize',8)
```

### 7.5.2.3 fontweight

Same as standard `fontweight` matlab option

```
>> plotmodel(md,'data',md.vel,'title','Ice velocity [m/yr]', 'fontweight','b')
```

### 7.5.2.4 xlabel

Same as standard `xlabel` matlab option

```
>> plotmodel(md,'data',md.vel,'xlabel','x axis [m]')
```

### 7.5.2.5 ylabel

Same as standard `ylabel` matlab option

```
>> plotmodel(md,'data',md.vel,'ylabel','y axis [m]')
```

### 7.5.2.6 text

Same as standard `text` matlab option

```
>> plotmodel(md,'data',md.vel,'text','Amendsen Sea','textposition',[2*10^3 3*10^4])
```

### 7.5.2.7 textposition

This option is used to place the text given in the `'text'` option. It contains the 2d or 3d coordinates of the text to be displayed.

```
>> plotmodel(md,'data',md.vel,'text','Amendsen Sea','textposition',[2*10^3 3*10^4])
```

### 7.5.2.8 textsize

Same as standard `fontsize` matlab option, applied to the text given in the `'text'` option.

```
>> plotmodel(md,'data',md.vel,'text','Amendsen Sea','textposition',[2*10^3 3*10^4], 'fontsize',8)
```

### 7.5.2.9 textweight

Same as standard `fontweight` matlab option, applied to the text given in the `'text'` option.

```
>> plotmodel(md,'data',md.vel,'text','Amendsen Sea','textposition',[2*10^3 3*10^4], 'textweight',8)
```

### 7.5.2.10 textcolor

Color of the text given in the 'text' option.

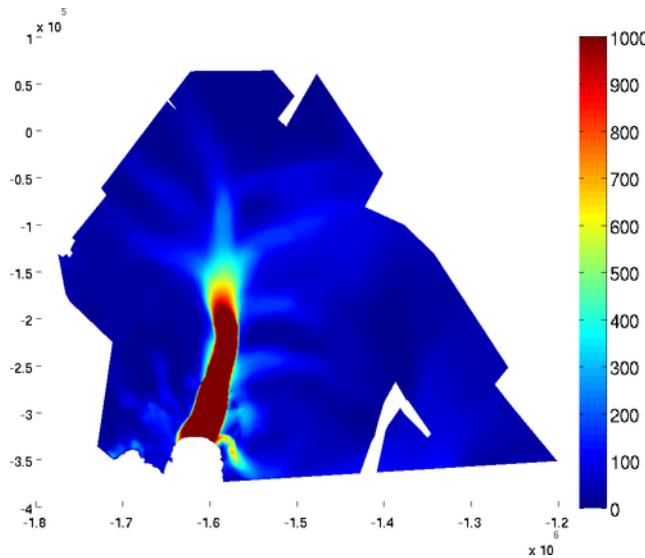
```
>> plotmodel(md,'data',md.vel,'text','Amendsen Sea','textposition',[2*10^3 3*10^4],'textcolor','b')
```

## 7.5.3 Color bars

### 7.5.3.1 caxis

Same as standard `caxis` matlab option (control the extreme values of the colorbar).

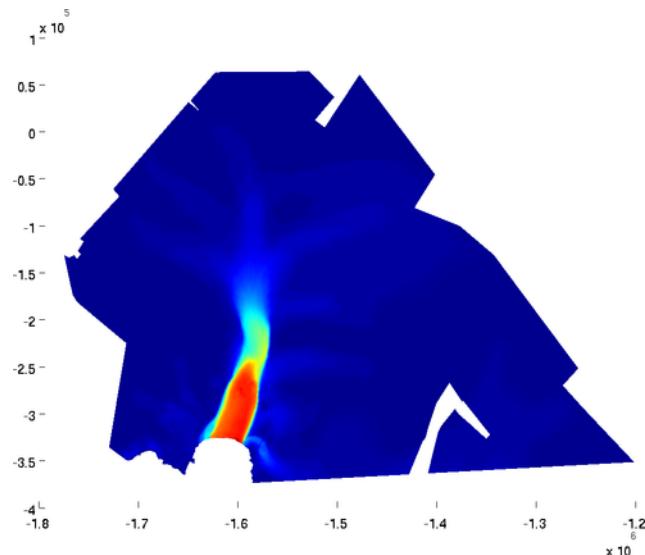
```
>> plotmodel(md,'data',md.vel,'caxis',[0 1000])
```



### 7.5.3.2 colorbar

This option is used to control the colorbar display: 'on' or 'off'.

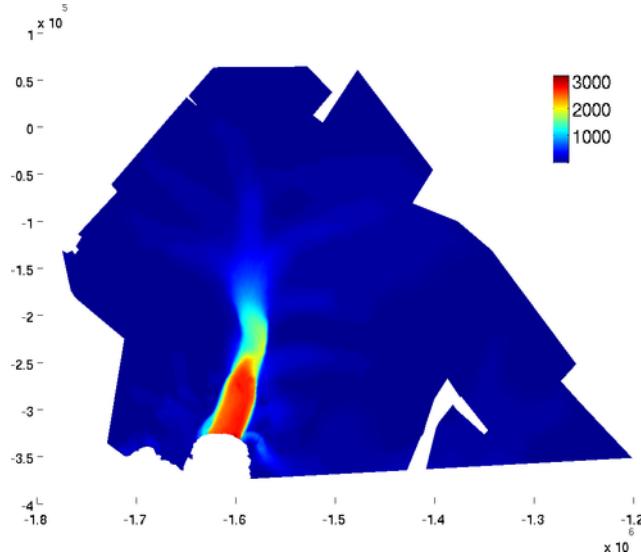
```
>> plotmodel(md,'data',md.vel,'colorbar','off')
```



### 7.5.3.3 colorbarpos

This option is used to control the colorbar position. It must be followed by a sequence [x\_position y\_position width length] in relative coordinates.

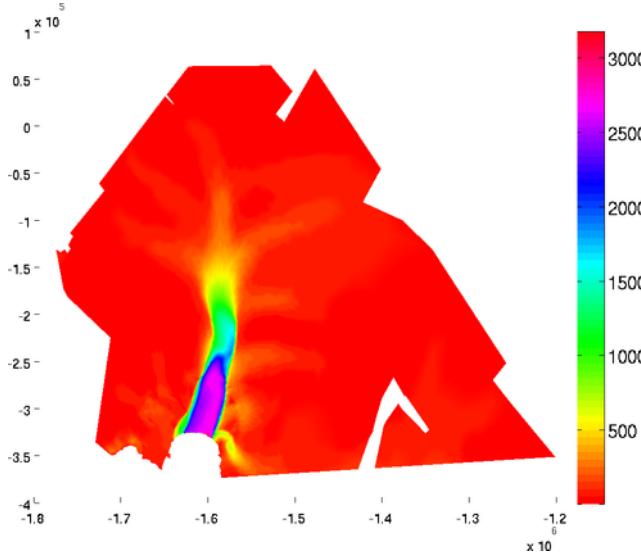
```
>> plotmodel(md,'data',md.vel,'colorbarpos',[0.80 0.70 0.02 0.15])
```



### 7.5.3.4 colormap

Same as standard [colormap](#) matlab option (control the extreme values of the colorbar).

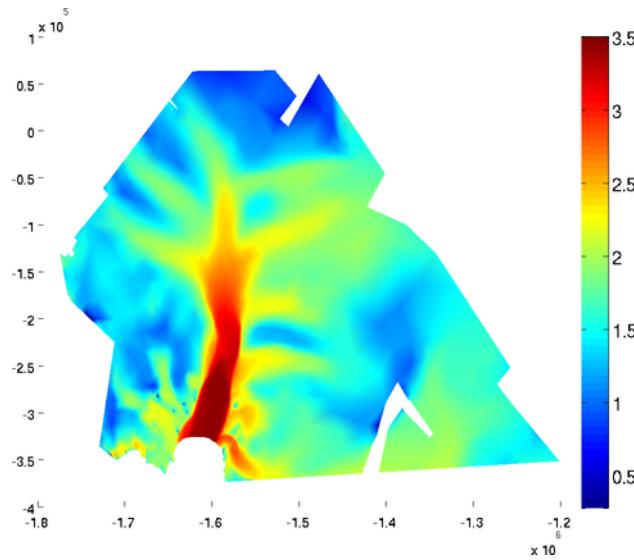
```
>> plotmodel(md,'data',md.vel,'colormap','hsv')
```



### 7.5.3.5 log

To get a logarithmic colorbar, use the 'log' option followed by 10 for a decimal logarithm.

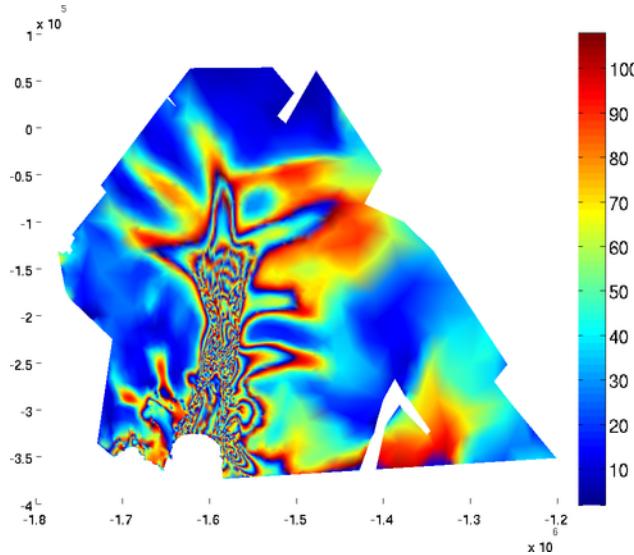
```
>> plotmodel(md,'data',md.vel,'log',10)
```



#### 7.5.3.6 wrapping

The option 'wrapping' is used to wrap the colormap. This option must be followed by the number of colorbar wrappings.

```
>> plotmodel(md,'data',md.vel,'wrapping',30)
```

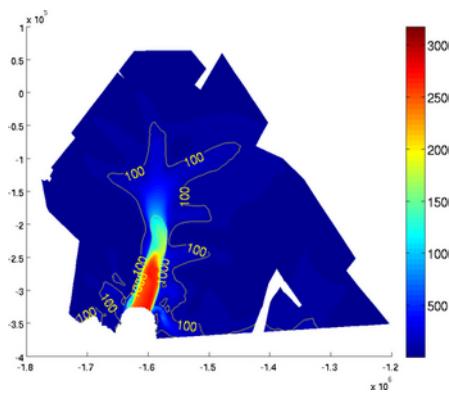


#### 7.5.4 Contours

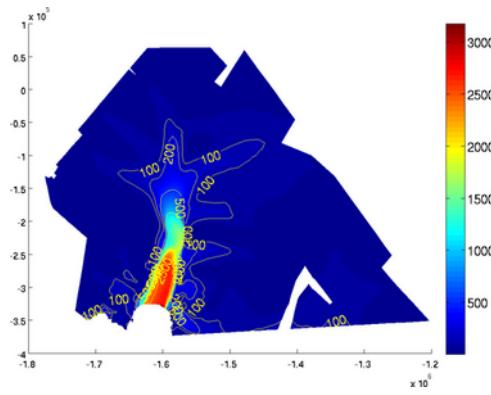
##### 7.5.4.1 contourlevels

Contours of equi-value can be added to the plot by using the 'contourlevels' option. The number of contours can be chosen by using the 'contourlevels' options. The user can specify a number of levels or a cell containing the values of color changes (See examples below).

```
>> plotmodel(md,'data',md.vel,'contourlevels',3)
```



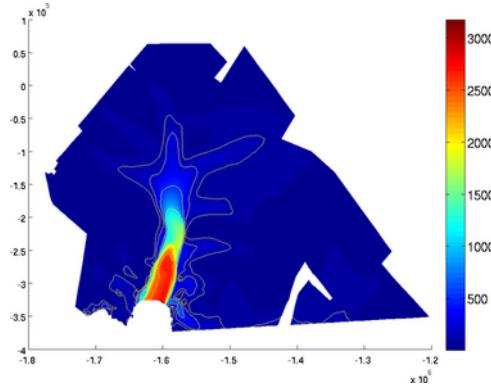
```
>> plotmodel(md,'data',md.vel,'contourlevels',[100,200,500,1000,2000,2500])
```



#### 7.5.4.2 contourticks

If the user does not want to display the contour levels ticks, use the 'contourticks' set as 'off':

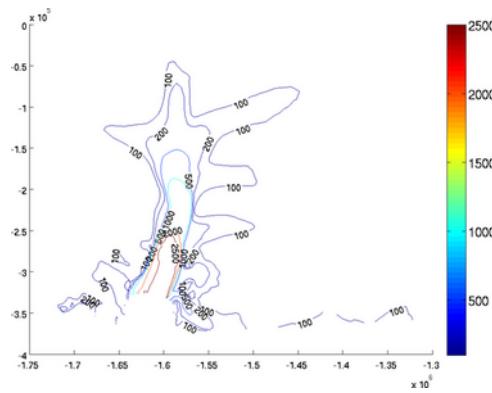
```
>> plotmodel(md,'data',md.vel,'contourlevels',[100,200,500,1000,2000,2500],'contourticks','off')
```



#### 7.5.4.3 contouronly

If the user wants to display the contours only, use the 'contouronly' set as 'on'.

```
>> plotmodel(md,'data','vel','contourlevels',[100,200,500,1000,2000,2500],'contouronly','on')
```



## 7.5.5 Latitude/Longitude

### 7.5.5.1 latlon

Latitude and longitude lines can be added to the figure by using the 'latlon' option. The argument can be 'on' or a cell array {latstep lonstep [resolution [color]]} where latstep, longstep and resolution are in degrees, color is a [r g b] array or a color string.

```
>> plotmodel(md,'data',md.vel,'latlon','on')

>> plotmodel(md,'data',md.vel,'latlon',{2 3 0.1 'r'})
```

### 7.5.5.2 latlonnumbering

Latitude and longitude labels can be added by using the 'latlonnumbering' option followed by 'on' or a cell array {latgap longap colordnumber latangle lonangle} where latgap and longap are pixel gaps for the numbers and the angles or used to rotate the text.

```
>> plotmodel(md,'data','vel','latlon',{2 3 0.1 'r'},'latlonnumbering',{3 3 'g' 90 0})
```

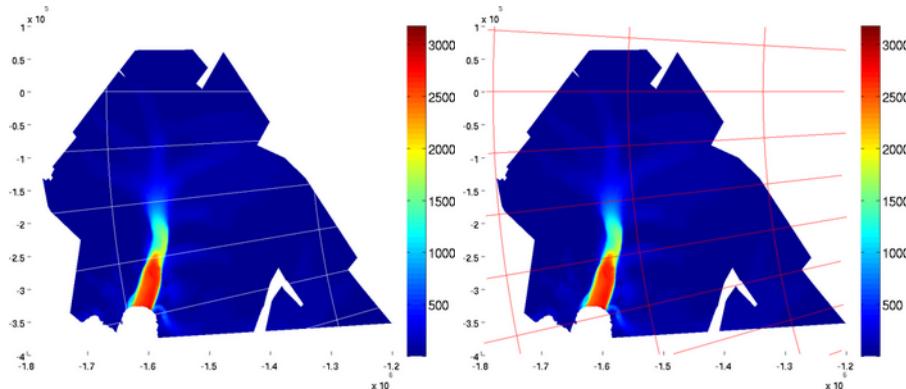


Figure 7.4: 'latlon' set as 'on' (left) and as a cell (right)

```
>> plotmodel(md,'data','vel','latlon',{2 3 0.1 'r'},'latlonnumbering','on')
```

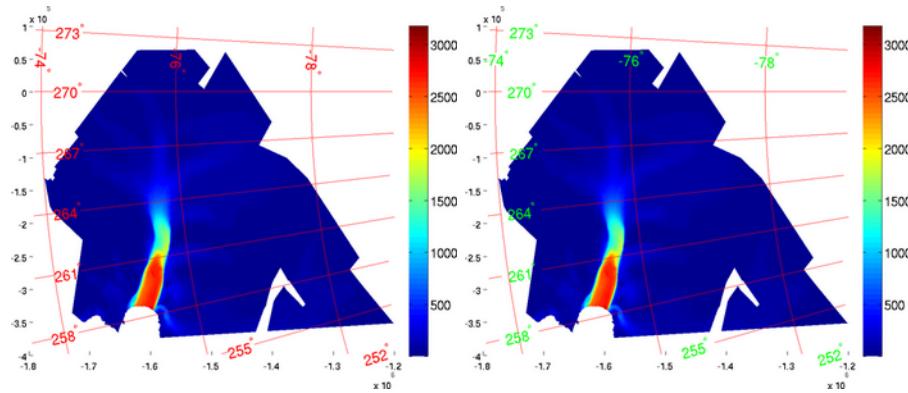


Figure 7.5: Here 'latlonnumbering' set as 'on' (left) and as a cell (right)

### 7.5.5.3 latlonclick

The labels can be manually placed with the 'latlonclick' option set as 'on'. The user will click on the location where he wants the label to be placed for each line.

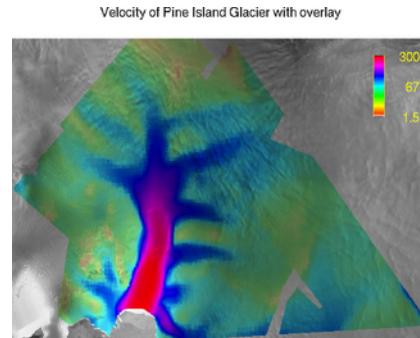
```
>> plotmodel(md,'data','vel','latlon',{2 3 0.1 'r'},'latlonnumbering',{3 3 'g' 90 0},'latlonclick',)
```

## 7.5.6 Radar overlay

### 7.5.6.1 overlay

The radar map of Antarctica can be superposed to the figure by using the 'overlay' option set as 'on' (gdal must be installed):

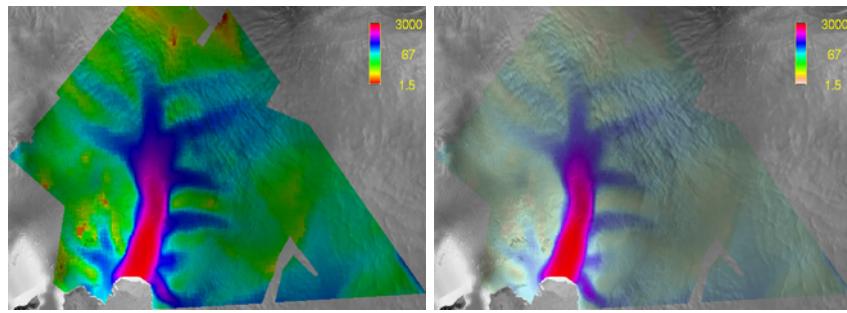
```
>> plotmodel(md,'data',md.vel,'overlay','on')
```



### 7.5.6.2 alpha

The size of each window can be changed with the 'alpha' option

```
>> plotmodel(md,'data',md.vel,'overlay','on','alpha',0.5)
>> plotmodel(md,'data',md.vel,'overlay','on','alpha',4.5)
```



### 7.5.6.3 highres

The radar image can be displayed in high resolution if the option 'highres' is set as 1

```
>> plotmodel(md,'data',md.vel,'overlay','on','highres',1)
```

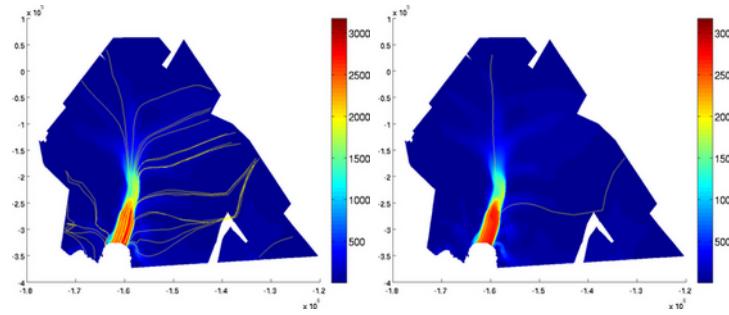
## 7.5.7 Other options

### 7.5.7.1 streamlines

Streamlines can be displayed by using the 'streamlines' option followed by a number of streamlines or a cell containing the coordinates of seed points:

```
>> plotmodel(md,'data',md.vel,'streamlines',50)
```

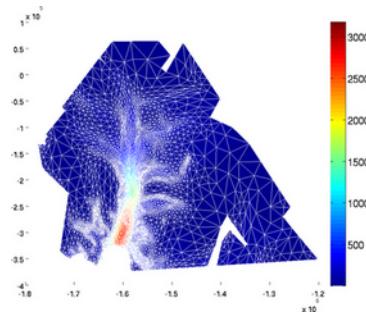
```
>> plotmodel(md,'data',md.vel,'streamlines',{10^6*[-1.45 -0.27],10^6*[-1.6 0]})
```



### 7.5.7.2 edgecolor

The mesh can be superposed to the plot by using the 'edgecolor' option followed by a color.

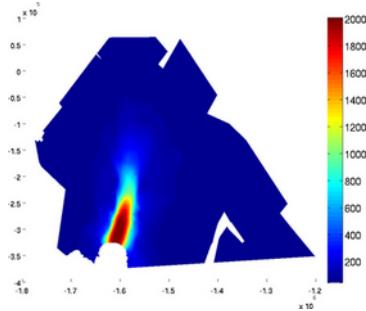
```
>> plotmodel(md,'data',md.vel,'edgecolor','w'})
```



### 7.5.7.3 smooth

The data can be smoothed by used the 'smooth' option followed by a number of averaging step. At each step, the field is averaged over each node as a weighed sum of the value of each element.

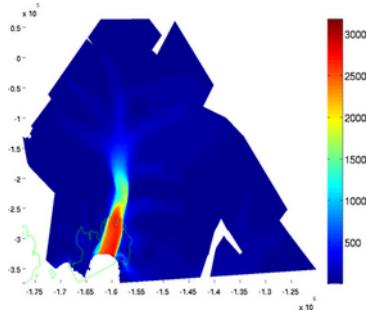
```
>> plotmodel(md,'data',md.vel,'smooth',200})
```



### 7.5.7.4 expdisp

Any Argus file can be displayed with the 'expdisp' option followed by the name of the Argus file.

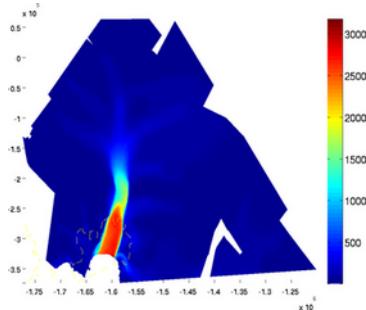
```
>> plotmodel(md,'data',md.vel,'expdisp','Iceshelves.exp')
```



### 7.5.7.5 expstyle

The style of the Argus profile can be controled with the 'expstyle' option, followed by the desired line style. Here is an example for a yellow dotted line:

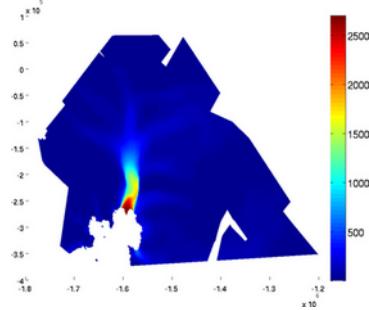
```
>> plotmodel(md,'data',md.vel,'expdisp','Iceshelves.exp','expstyle','--y')
```



### 7.5.7.6 mask

If one does not want to display the value of the field on a mask only, use the 'mask' option followed by a vector that holds 0 for the vertices whose values are hidden:

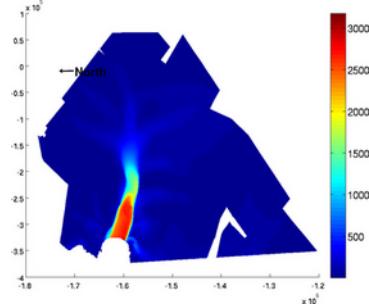
```
>> plotmodel(md,'data',md.vel,'mask',md.geography.vertexongroundedice)
```



### 7.5.7.7 northarrow

An arrow pointing North can be added with the 'northarrow' option followed by 'on'. The shape and position of the arrow can be controled by using [x0 y0 length [ratio [width]]] instead of 'on'.

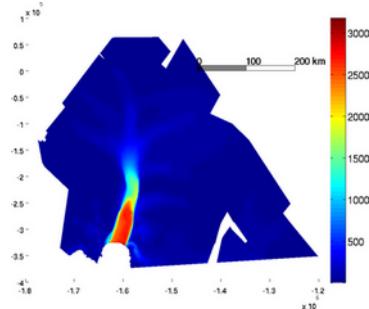
```
>> plotmodel(md,'data',md.vel,'northarrow','on')}
```



### 7.5.7.8 scalerruler

A scale ruler can be added. As for the North arrow, the default display is done by 'on' but the shape and position of the scale ruler can be controled by [x0 y0 length width numberofticks] where (x0,y0) are the coordinates of the lower left corner.

```
>> plotmodel(md,'data',md.vel,'scaleruler','on')}
```



# Chapter 8

## Miscellaneous Tools

Several tools are available to help the user analyse the results and set up the models. These tools are just mentionned here, interested user can learn how to use these tools by typing `help function_name` in the Matlab prompt for any of the following functions.

### 8.0.8 Mesh

- `setmesh` generate a mesh from a domain outline
- `meshexprefine` refine a region of a mesh
- `meshprocessrift` process mesh when rifts are present
- `MeshQuality` compute mesh quality
- `meshyams` anysotropic mesh generation
- `rifftiprefine` refine mesh near rift tips

### 8.0.9 Model parameterization

- `extrude` vertically extrude a model
- `setmask` establish boundaries between grounded and floating ice
- `modelextract` extract the model over a subdomain
- `parameterize` model general parameterization
- `parametercontrolB` set default parameters for a control method on B
- `parametercontroldrag` set default parameters for a control method on drag
- `plugvelocities` plug observed velocities
- `plugvelocitiesraw` plug raw observed velocities
- `setflowequation` set diagnostic elements type
- `solversettoasm` set PETSc solver to ASM
- `solversettomumps` set PETSc solver to MUMPS
- `solversettosor` set PETSc solver to SOR
- `ThicknessCorrection` create a transition between the thickness on the grounding line and hydrostatic equilibrium
- `SetIceSheetBC` set ice sheet boundary conditions
- `SetIceShelfBC` set ice shelf boundary conditions
- `SetMarineIceSheetBC` set marine ice sheet boundary conditions

### 8.0.10 Mask

- `contourenvelope` create a list of segments enveloping an Argus contour
- `ContourToMesh` get elements and/or nodes inside an Argus contour
- `GetAreas` compute the area of each element
- `xy2ll` convert lat/lon to (x,y)
- `ll2xy` convert (x,y) to lat/lon
- `utm2ll` convert UTM to lat/lon

### 8.0.11 Interpolation

- `InterpFromGridToMesh` interpolation from a grid to a list of (x,y)
- `InterpFromMeshToGrid` interpolation from a 2d mesh to a grid
- `InterpFromMeshToMesh2d` interpolation from a 2d mesh to a list of (x,y)
- `InterpFromMeshToMesh2d` interpolation from a grid to a list of (x,y)

### 8.0.12 Argus files

- `expcoarsen` coarsen or refine the resolution a contour
- `exptools` create and manage Argus files
- `pread` read an Argus file
- `pwrt` write an Argus file

### 8.0.13 Results analysis

- `averaging` data averaging over a mesh
- `basalstress` compute the basal stress
- `contourmassbalance` compute the mass balance of a contour
- `DepthAverage` depth averaging of a 3d field
- `drivingstress` compute the driving stress
- `flowlines` compute the coordinates of one or several flowlines
- `paterson` compute B from a temperature
- `project2d` project a 3d field on a layer
- `project3d` extrude a 2d field on every layer
- `SectionValues` compute the value of a field on a section or line
- `thicknessesevolution` compute dh/dt

# Chapter 9

## Tutorial

### 9.1 Pine Island Glacier tutorial

This section gives the user an example of simulation using `gdal` and ISSM to compute the velocity field of Pine Island Glacier (West Antarctica).

### 9.2 Radar image

In order to build a proper domain outline, one needs an image of the glacier. This image is often a radar image that needs to be resized. The tools for accomplishing this feat are `gdal_translate` and `gdalwarp`, both part of the `gdal` suite of utilities. For this example we will be using the Antarctica mosaic LIMA, available on <http://lima.nasa.gov/>. Here is the original radar image:

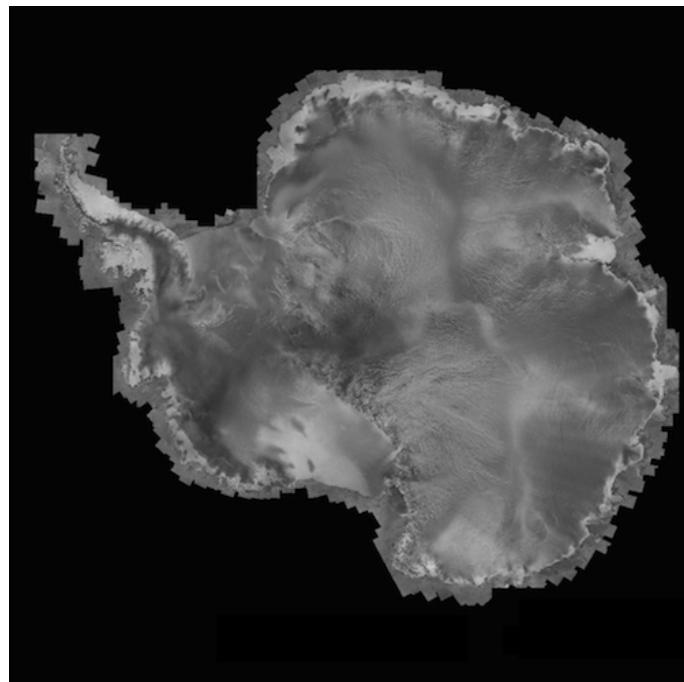


Figure 9.1: Original radar image

To display the original mosaic properties, type:

```
$ gdalinfo file.gif
```

where `file.gif` is the name of this mosaic. At the end of the properties, the coordinates of the corners are displayed:

```
Upper Left  (-3174450.000, 2406320.000) ( 52d50'12.76"W, 54d28'32.46"S)
Lower Left  (-3174450.000,-2815680.000) (131d34'20.81"W, 52d18'43.35"S)
Upper Right ( 2867550.000, 2406320.000) ( 49d59'53.25"E, 56d29'48.10"S)
Lower Right ( 2867550.000,-2815680.000) (134d28'37.50"E, 54d10'45.39"S)
Center      (-153450.000, -204680.000) (143d 8'27.07"W, 87d38'45.21"S)
```

The first step is to determine the coordinates for the box of interest. In this case we will just pull out Pine Island Glacier. The bounding box is approximately -1800000 100000 and -1200000 -400000 (upper left to lower right). To extract Pig, we use:

```
$ gdal_translate -a_ullr -1800000 100000 -1200000 -400000 -projwin -1800000 100000 -1200000 -400000 f
```

The `-projwin` option specifies the area one wants to clip out of the mosaic using the coordinate system. We also used the `-a_ullr` option to force the output image to have the bounding coordinates we want. The result is:

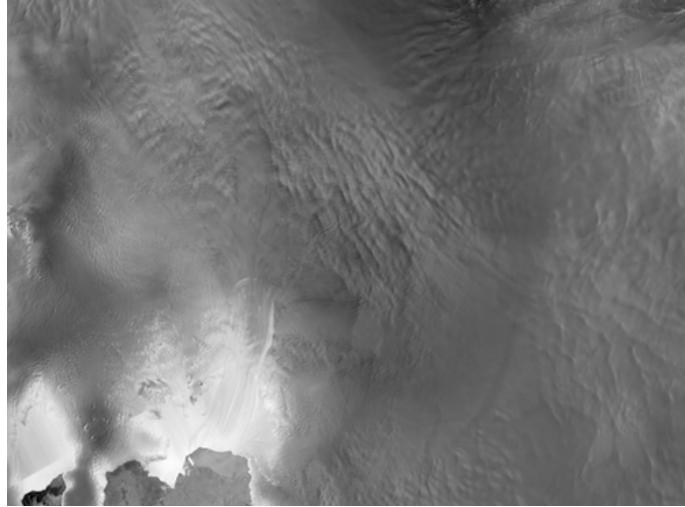


Figure 9.2: Pine Island Glacier radar image

## 9.3 Mesh creation

### 9.3.1 Creation of the domain outline

To create the mesh, one needs a domain outline. We will use the extracted radar image to draw the domain outline using `expmaster` that will generate an Argus file that holds all the points of the contour. The first step is to display the image with matlab, while keeping the coordinates. To do so, use the following commands:

```
>> I=imread('pig.tif'); %store the image matrix in I
>> x=linspace(-1800000,-1200000,size(I,2)); %reconstruct x
>> y=linspace(100000,-400000,size(I,1)); %reconstruct y
>> imagesc(x,y,I);colormap('gray');set(gca,'Ydir','normal')%display image in gray scale
```

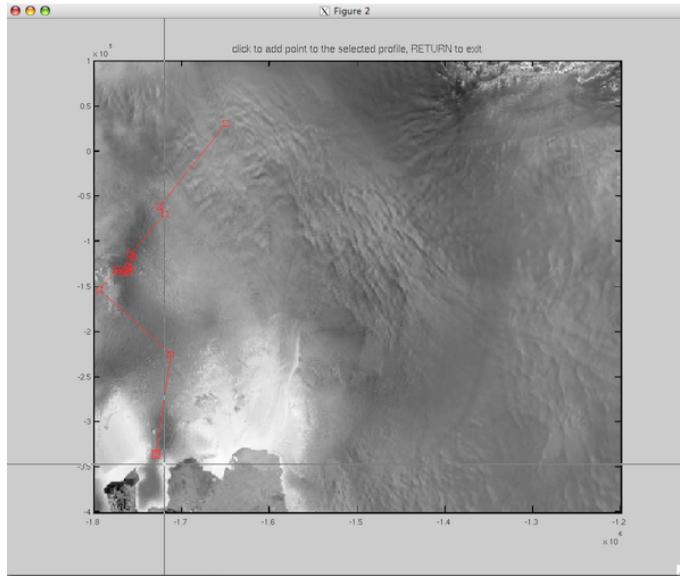
Now that the radar image is displayed, we can call `expmaster` to draw the domain outline. If the domain outline is called `DomainOutline.exp`, type:

```
>> expmaster('DomainOutline.exp')
```

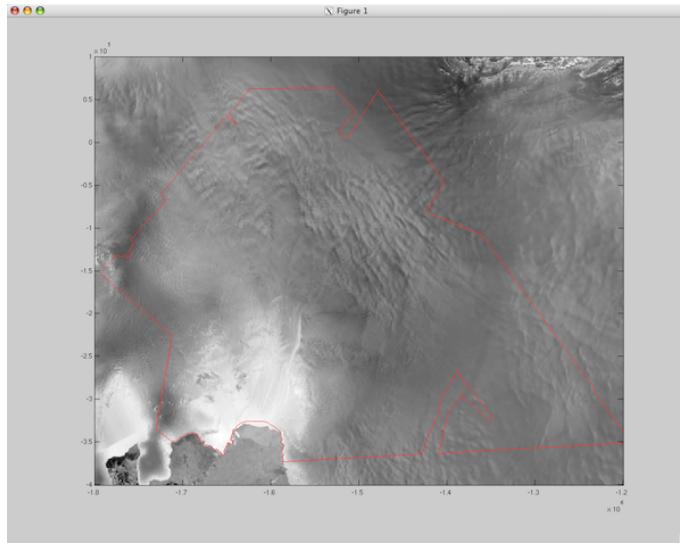
Then click on *create a contour (closed)*:



You can now click on the image to create the contour. Once you are done, press Enter. Do not hesitate to use the expmaster options such as undo, add point,...



The domain outline should look like that:



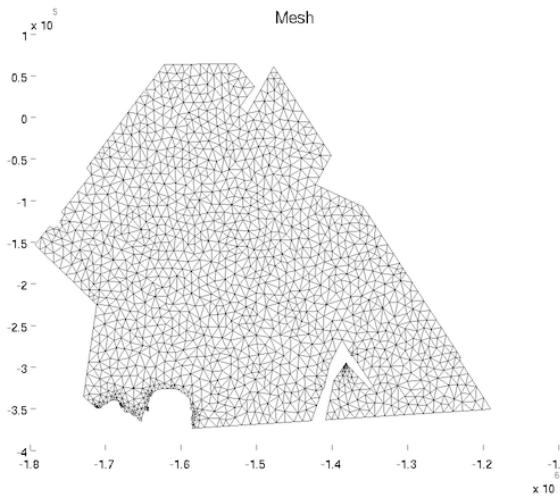
### 9.3.2 Mesh generation

You can now mesh the domain using the `mesh` routine. The resolution (in meters) is here specified as 10000m:

```
>> md=model;
>> md=mesh(md, 'DomainOutline.exp', 10000);
```

To display the mesh, use `plotmodel`:

```
>> plotmodel(md,'data','mesh')
```

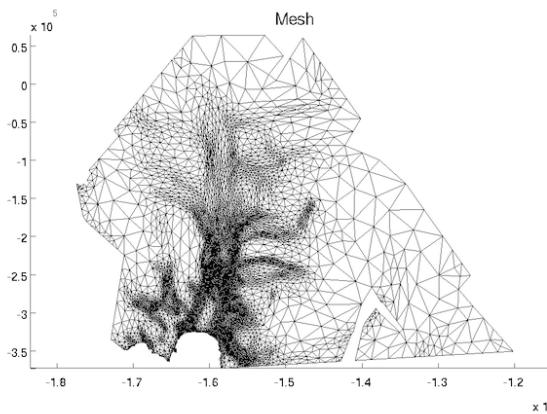


### 9.3.3 Anisotropic mesh adaptation

If the user dispose of a velocity map, it can be used to generate an adapted mesh by using [YAMS](#) developed by [Pascal Frey](#). We apply it on Pine Island with the following options:

```
nsteps=4;
velocitypath='/u/wilkes-r1b/larour/ModelData/RignotPigVel1996/RignotPig1996_HoleFree.mat';
domainoutline='DomainOutline.exp';
gradation=3*ones(nsteps,1);
epsilon=2.5;
resolution=2000;
hmin=1500;           %1.5km
hmax=80*10^3;        %80 km

md=model;
md=meshyams(md,'domainoutline',domainoutline,'velocities',velocitypath,'nsteps',nsteps, ...
'gradation',gradation,'resolution',resolution,'hmin',hmin,'hmax',hmax,'epsilon',epsilon);
```



## 9.4 Geography

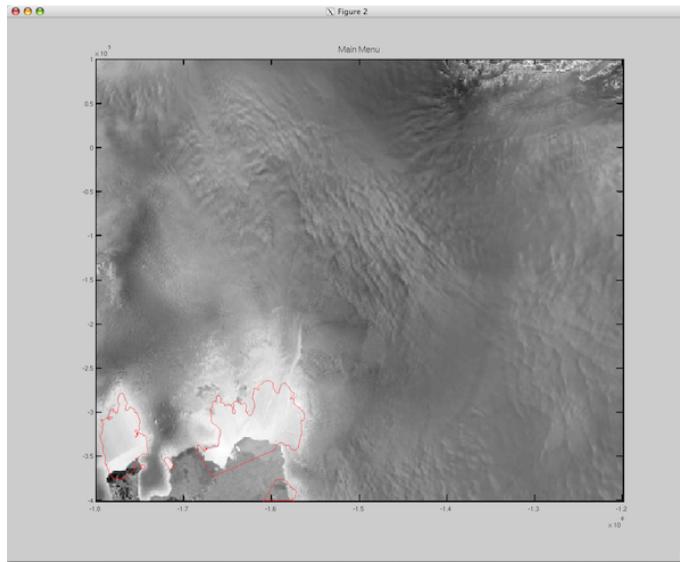
Two Argus files must be created:

- `Iceshelves.exp` a file that holds the contours of the ice shelves
- `Island.exp` a file that holds the contours of the grounded ice included in the ice shelves (ice rises, islands,...)

### 9.4.1 Building the 'Iceshelves.exp' file

We will use the procedure as the one to create the domain outline. The position of the grounding line must be known. The contour must be closed and include all the ice shelves.

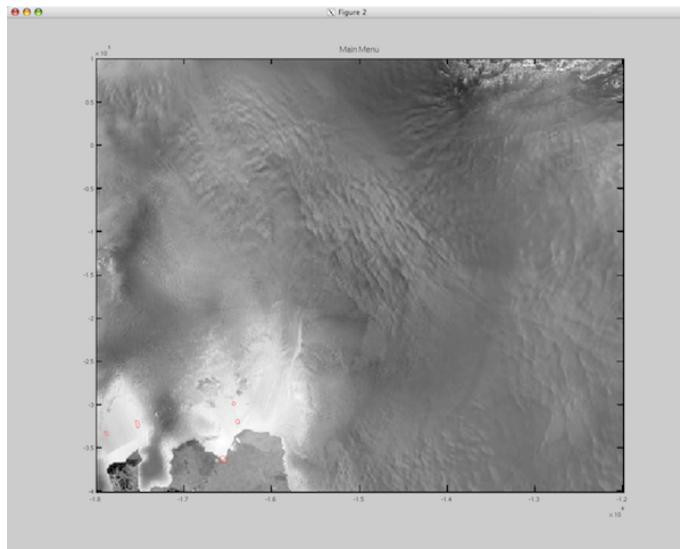
```
>> I=imread('pig.tif'); %store the image matrix in I
>> x=linspace(-1800000,-1200000,size(I,2)); %reconstruct x
>> y=linspace(100000,-400000,size(I,1)); %reconstruct y
>> imagesc(x,y,I);colormap('gray');set(gca,'Ydir','normal')%display image in gray scale
>> expmaster('Iceshelves.exp'); %call expmaster
```



### 9.4.2 Building the 'Island.exp' file

Repeat the previous step and create the contours of the islands in the ice shelf.

```
>> imagesc(x,y,I);colormap('gray');set(gca,'Ydir','normal')%display image in gray scale
>> expmaster('Islands.exp'); %call expmaster
```



#### 9.4.3 Geography command

To set the model geography, use the following command:

```
>> md=geography(md,'Iceshelves.exp','Islands.exp');
```

### 9.5 Parameter file

Here is just an example of parameter file called `Pig.par`. Several data are needed (surface, thickness, surface temperature,...). The content of these files is not detailed here and the user will have to change the parts of the parameter file that refer to these data to fit his.

```
%%%%%%%%%%%%%%%%Some hardcoded parameters for this deck%%%%%%%
%some parameterization for this parameter file :)
modelldatapath='/u/astrid-r1b/larour/ModelData';
thicknesspath=[modelldatapath '/BedMap/gridded/thickness.mat'];
firnpath=[modelldatapath '/BroekeFirn1km/firn.mat'];
surfacepath=[modelldatapath '/BamberDEMAntarctica1km/surface_smooth30_lowslope.mat'];
thicknessiceshelvespath=[modelldatapath '/HartmutThicknessAntarctica/HartmutThickness.mat'];
mosaicpath=[modelldatapath '/RignotAntarcticaVelMosaicRampErsAlos/RignotAntVel.mat'];
temperaturepath=[modelldatapath '/GiovinettoZwallyTemperatures92/Giovinetto_Temperatures.mat'];
heatfluxpath=[modelldatapath '/HeatfluxAntarctica/RignotHeatFlux.mat'];

%Cluster options
md.cluster=astrid('np',3);
%%%%%%%%%%%%%%%%
disp('      reading bedmap thicknesses');
md.thickness=InterpFromFile(md.x,md.y,thicknesspath,10);

disp('      reading firn layer');
md.firn_layer=InterpFromFile(md.x,md.y,firnpath,0);

disp('      reading Bamber dem');
md.surface=InterpFromFile(md.x,md.y,surfacepath,10);

%correct the surface by taking into account the firn layer
```

```

rho_ice=917;
rho_firn=830;
firn_layer_correction=md.firn_layer*(1-rho_firn/rho_ice);
md.surface=md.surface-firn_layer_correction;

%Correct thickness by assuming hydrostatic equilibrium 10km down the grounding line
md=ThicknessCorrection(md);

disp('      reading velocities from Rignot');
md=plugvelocities(md,mosaicpath,0);

%Take care of drag
md.drag_type=2; %0 none 1 plastic 2 viscous
md.drag_coefficient=300*ones(md.numberofgrids,1); %q=1.

%Take care of iceshelves: no drag md.drag
pos=find(md.elementoniceshelf);
md.drag_coefficient(md.elements(pos,:))=0;
md.p=ones(md.numberofelements,1);
md.q=ones(md.numberofelements,1);

%flow law
disp('      creating flow law paramters');
md.rheology_n=3*ones(md.numberofelements,1);
md.rheology_B=paterson(md.observed_temperature);

disp('      creating accumulation rates');
md.accumulation_rate=ones(md.numberofgrids,1); %1m/a

%Deal with boundary conditions:
disp('      boundary conditions');
md=SetMarineIceSheetBC(md);

disp('      thermal model');
md.observed_temperature=InterpFromFile(md.x,md.y,temperaturepath,253);
md.melting_rate=zeros(md.numberofgrids,1);
md.observed_temperature=md.temperature;

disp('      reading geothermal flux');
load(heatfluxpath);
md.geothermalflux=InterpFromGridToMesh(x_m,y_m,heatflux_Antarctica,md.x,md.y,80);
md.geothermalflux=md.geothermalflux/1000; %map is given in mW/m^2, we need it in W/m^2
%%%%%%%

```

### 9.5.1 Parameterization command

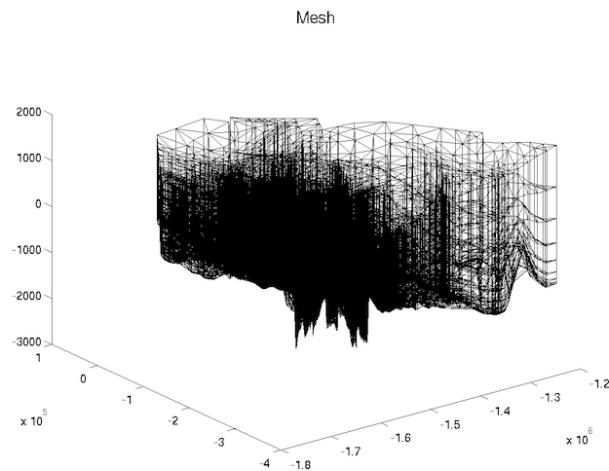
To parameterize the model, use the following command:

```
>> md=parameterize(md,'Pig.par');
```

### 9.5.2 Extrusion (optional)

If the user wants to use a 3d model (Pattyn or Stokes), the model must now be extruded. In the following example, the model is extruded in 10 layers with an extrusion exponent of 3.

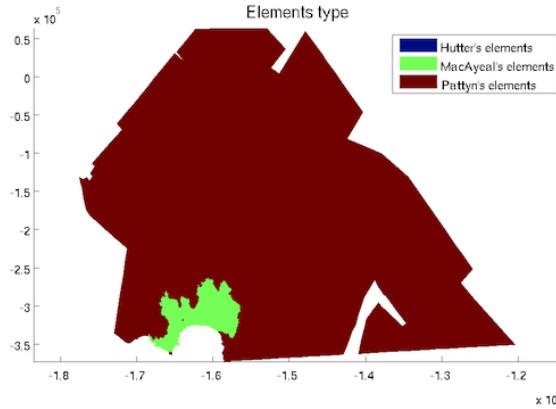
```
>> md=extrude(md,10,3);
```



### 9.5.3 Setting elements type

Last parameterization step: assigning the models to the elements (here, MacAyeal on the ice shelf and Pattyn for the grounded ice).

```
>> md=setelementstype(md,'MacAyeal',md.elementoniceshelf,'fill','Pattyn');
>> plotmodel(md,'data','elements_type','view',2)
```



## 9.6 Wrap up

So far, we have used the following commands to create the model:

```
>> md=model; %create an empty model structure
>> md=mesh(md,'DomainOutline.exp',10000); %mesh generation
>> md=geography(md,'Iceshelves.exp','Islands.exp'); %geography
>> md=parameterize(md,'Pig.par'); %parameterization
>> md=extrude(md,10,3); %3d extrusion
>> md=setelementstype(md,'MacAyeal',md.elementoniceshelf,'fill','Pattyn'); %elements type setting
```

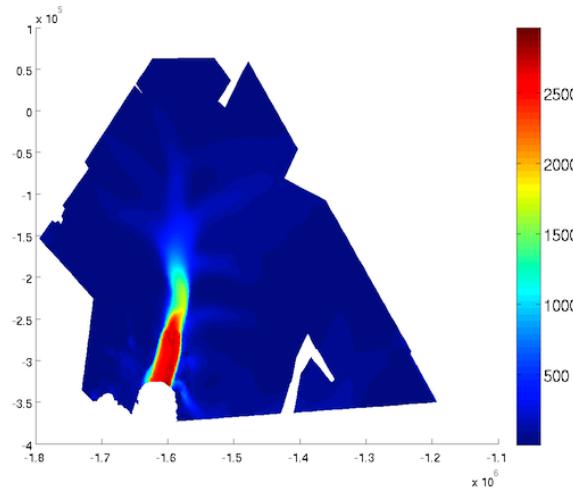
## 9.7 Launching the diagnostic solution sequence

The model is now ready to be used in a diagnostic solution sequence.

```
>> md=solve(md,DiagnosticSolutionEnum);
```

the velocity field is now ready to be displayed:

```
>> plotmodel(md,'data',md.results.DiagnosticSolution.Vel)
```



# Chapter 10

## FAQ

### 10.1 The following message appears in the errlog file when launching my job in parallel:

```
mpdrun_wilkes.jpl.nasa.gov: cannot connect to local mpd (/tmp/mpd2.console_larour); possible causes:  
1. no mpd is running on this host  
2. an mpd is running but was started without a "console" (-n option)  
~  
~
```

This message means that the MPI (Message Passing Interface) server, called mpd, is not running. Therefore, no parallel jobs can run on the cluster. To solve this issue, just type, at the command prompt on the server side (if for example your cluster has 8 cpus):

```
mpd --ncpus=8 &
```

This will launch the MPI server to manage 8 cpus on the cluster.

### 10.2 How to debug a matlab crash in serial mode?

There is no better way to debug than using [valgrind](#). Install [valgrind](#) in the directory \$ISSM\_TIER/externalpackages/valg and use the following command:

```
matlab -nojvm -nosplash -r "matlab_function(params)" -D"valgrind --error-limit=no --tool=memcheck -v
```

Valgrind output valgrind.log should help.

### 10.3 How to debug a parallel solution crash?

Again, [valgrind](#) is the perfect tool and is embedded in ISSM. Set the model debugging field to 1 and use only one cpu:

```
md.mem_debug=1;  
md.cluster.np=1;
```

Launch the solution sequence and read the 'errlog' file. [valgrind](#) is also very helpful to track memory leaks.